ELE HSM API Rev 1.0

Generated by Doxygen 1.8.17

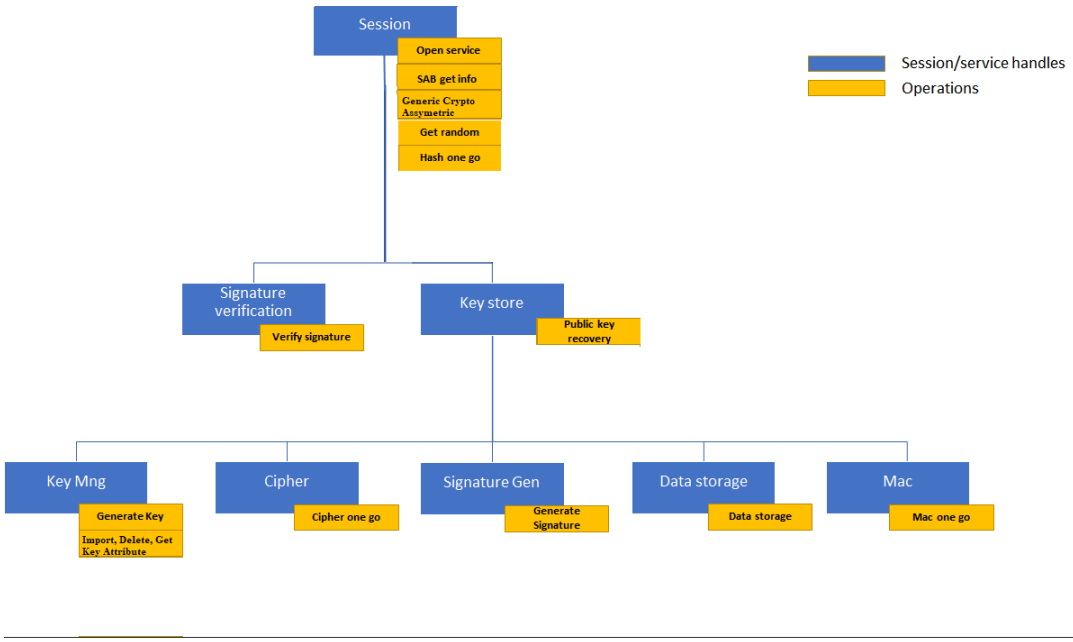
1 ELE HSM API

This document is a software referece description of the API provided by the i.MX8ULP, i.MX93 HSM solutions for ELE Platform.

2 Revision History

Revision	date	description
0.1	Apr 27 2023	Preliminary draft

3 General concepts related to the API



3.1 Session

The API must be initialized by a potential requestor by opening a session.
The session establishes a route (MU, DomainID...) between the requester and the HSM. When a session is opened, the HSM returns a handle identifying the session to the requester.

3.2 Service flow

For a given category of services which require service handle, the requestor is expected to open a service flow by invoking the appropriate HSM API.
The session handle, as well as the control data needed for the service flow, are provided as parameters of the call.
Upon reception of the open request, the HSM allocates a context in which the session handle,as well as the provided control parameters are stored and return a handle identifying the service flow.
The context is preserved until the service flow, or the session, are closed by the user and it is used by the HSM to proceed with the sub-sequent operations requested by the user on the service flow.

3.3 Example

```
/* Open a session: create a route between the user and the HSM */
hsm_open_session(&open_session_args, &session_hdl);

/* Open a key store - user is authenticated */
hsm_open_key_store_service(session_hdl, &open_svc_key_store_args, &key_store_hdl);


/* Open cipher service - it grants access to ciphering operations */
hsm_open_cipher_service(key_store_hdl, &open_svc_cipher_args, &cipher_hdl);

/*    Perform  ECB, CCB ... */
hsm_cipher_one_go (cipher_hdl, &op_cipher_one_go_args);
/* Perform authenticate and encryption algos: e.g GCM */
hsm_auth_enc (cipher_hdl, &op_auth_enc_args);
/* Perform hashing operations: e.g SHA */
hsm_hash_one_go (hash_hdl, &op_hash_one_go_args);

/* Close the session and all the related services */
hsm_close_session(session_hdl);
```

3.4 Key store

A key store can be created by specifying the CREATE flag in the hsm_open_key_store_service API. Please note that the created key store will be not stored in the NVM till a key is generated or imported specyfing the "STRICT OPERATION" flag.
Only symmetric and private keys are stored into the key store. Public keys can be exported during the key pair generation operation or recalculated through the hsm_pub_key_recovery API.
Secret keys cannot be exported under any circumstances, while they can be imported in encrypted form.

3.4.1 Key management

Keys are divided in groups, keys belonging to the same group are written/read from the NVM as a monolitic block.
Up to 3 key groups can be handled in the HSM local memory (those immediately available to perform crypto operations), while up to 1000 key groups can be handled in the external NVM and imported in the local memory as needed.
If the local memory is full (3 key groups already reside in the HSM local memory) and a new key group is needed by an incoming user request, the HSM swaps one of the local key group with the one needed by the user request.
The user can control which key group must be kept in the local memory (cached) through the manage_key_group API lock/unlock mechanism.
As general concept, frequently used keys should be kept, when possible, in the same key group and locked in the local memory for performance optimization.

3.4.2 NVM writing

All the APIs creating a key store (open key store API) or modyfing its content (key generation, key_management, key derivation functions) provide a "STRICT OPERATION" flag. If the flag is set, the HSM exports the relevant key store blocks into the external NVM and increments (blows one bit) the OTP monotonic counter used as roll back protection. In case of key generation/derivation /update the "STRICT OPERATION" has effect only on the target key group.
Any update to the key store must be considered as effective only after an operation specifying flag "STRICT OP↩ ERATION" is aknowledged by the HSM. All the operations not specifying the "STRICT OPERATION" flags impact the HSM local memory only and will be lost in case of system reset
Due to the limited monotonic counter size, the user should, when possible, perform multiple udate before setting the "STRICT OPERATION" flag(i.e. keys to be updated should be kept in the same key group).
Once the monotonic counter is completely blown a warning is returned on each key store export to the NVM to inform the user that the new updates are not roll-back protected.

3.5 Implementation specificities

HSM API with common features are supported on i.MX8ULP and i.MX93.The details of supported features per chip will be listed in the platform specifities.

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Module Documentation

5.1 Session

The API must be initialized by a potential requestor by opening a session.
Once a session is closed all the associated service flows are closed by the HSM.

Data Structures

- struct hsm_session_hdl_s
- struct hsm_service_hdl_s
- struct open_session_args_t

Macros

- #define HSM_MAX_SESSIONS (8u)

  Maximum sessions supported.
- #define HSM_MAX_SERVICES (32u)

  Maximum services supported.
- #define HSM_OPEN_SESSION_PRIORITY_LOW (0x00U)

  Low priority. default setting on platforms that doesn't support sessions priorities.
- #define HSM_OPEN_SESSION_PRIORITY_HIGH (0x01U)

  High Priority session.
- #define HSM_OPEN_SESSION_FIPS_MODE_MASK (1u << 0)

  Only FIPS certified operations authorized in this session.
- #define HSM_OPEN_SESSION_EXCLUSIVE_MASK (1u << 1)

  No other HSM session will be authorized on the same security enclave.
- #define HSM_OPEN_SESSION_LOW_LATENCY_MASK (1u << 3)

  Use a low latency HSM implementation.
- #define HSM_OPEN_SESSION_NO_KEY_STORE_MASK (1u << 4)

  No key store will be attached to this session. May provide better performances on some operation depending on the implementation. Usage of the session will be restricted to operations that doesn't involve secret keys (e.g. hash, signature verification, random generation).
- #define HSM_OPEN_SESSION_RESERVED_MASK ((1u << 2) | (1u << 5) | (1u << 6) | (1u << 7))

  Bits reserved for future use. Should be set to 0.

Typedefs

- typedef uint32_t hsm_hdl_t

Functions

- hsm_err_t hsm_open_session (open_session_args_t *args, hsm_hdl_t *session_hdl)
- hsm_err_t hsm_close_session (hsm_hdl_t session_hdl)
- struct hsm_session_hdl_s * session_hdl_to_ptr (uint32_t hdl)
- struct hsm_service_hdl_s * service_hdl_to_ptr (uint32_t hdl)
- void delete_session (struct hsm_session_hdl_s *s_ptr)
- void delete_service (struct hsm_service_hdl_s *s_ptr)
- struct hsm_session_hdl_s * add_session (void)
- struct hsm_service_hdl_s * add_service (struct hsm_session_hdl_s *session)

5.1.1 Detailed Description

The API must be initialized by a potential requestor by opening a session.
Once a session is closed all the associated service flows are closed by the HSM.

5.1.2 Data Structure Documentation

5.1.2.1 struct hsm_session_hdl_s Structure describing the session handle members

Data Fields

struct plat_os_abs_hdl *	phdl	Pointer to OS device node.
uint32_t	session_hdl	Session handle.
uint32_t	mu_type	Session MU type.

5.1.2.2 struct hsm_service_hdl_s Structure describing the service handle members

Data Fields

struct hsm_session_hdl_s *	session	Pointer to session handle.
uint32_t	service_hdl	Service handle.

5.1.2.3 struct open_session_args_t Structure detailing the open session operation member arguments

Data Fields

uint32_t	session_hdl	Session handle.
uint8_t	session_priority	Priority of the operations performed in this session.
uint8_t	operating_mode	Options for the session to be opened (bitfield).
uint8_t	interrupt_idx	Interrupt number of the MU used to indicate data availability.

5.1.3 Typedef Documentation

5.1.3.1 hsm_hdl_t `typedef uint32_t hsm_hdl_t`

Define the HSM handle type

5.1.4 Function Documentation

5.1.4.1 hsm_open_session() `hsm_err_t hsm_open_session ( open_session_args_t * args, hsm_hdl_t * session_hdl )`

Parameters

args	pointer to the structure containing the function arguments.
session_hdl	pointer to where the session handle must be written.

Returns

error_code error code.

5.1.4.2 hsm_close_session() `hsm_err_t hsm_close_session ( hsm_hdl_t session_hdl )`

Terminate a previously opened session. All the services opened under this session are closed as well

Parameters

session_hdl	pointer to the handle identifying the session to be closed.

Returns

error_code error code.

5.1.4.3 session_hdl_to_ptr() `struct hsm_session_hdl_s* session_hdl_to_ptr ( uint32_t hdl )`

Returns pointer to the session handle

Parameters

hdl	identifying the session handle.

Returns

pointer to the session handle.

5.1.4.4 service_hdl_to_ptr() `struct hsm_service_hdl_s* service_hdl_to_ptr ( uint32_t hdl )`

Returns pointer to the service handle

Parameters

hdl	identifying the session handle.

Returns

pointer to the service handle.

5.1.4.5 delete_session() `void delete_session ( struct hsm_session_hdl_s * s_ptr )`

Delete the session

Parameters

s_ptr	pointer identifying the session.

5.1.4.6 delete_service() `void delete_service ( struct hsm_service_hdl_s * s_ptr )`

Delete the service

Parameters

s_ptr	pointer identifying the service.

5.1.4.7 add_session() `struct hsm_session_hdl_s* add_session ( void )`

Add the session

Returns

pointer to the session.

5.1.4.8 add_service() `struct hsm_service_hdl_s* add_service ( struct hsm_session_hdl_s * session )`

Add the service

Returns

pointer to the service.

5.2 Key management

Data Structures

- struct op_delete_key_args_t
- struct op_get_key_attr_args_t
- struct op_import_key_args_t
- struct kek_enc_key_hdr_t
- struct op_generate_key_ext_args_t
- struct op_generate_key_args_t
- struct open_svc_key_management_args_t
- struct op_manage_key_group_args_t

Macros

- #define HSM_OP_DEL_KEY_FLAGS_STRICT_OPERATION ((hsm_op_import_key_flags_t)(1u << 7))
- #define HSM_OP_IMPORT_KEY_INPUT_E2GO_TLV ((hsm_op_import_key_flags_t)(1u << 0))

  Bit 0: set 1 means input is E2GO_TLV.
- #define HSM_OP_IMPORT_KEY_INPUT_SIGNED_MSG ((hsm_op_import_key_flags_t)(0u << 0))

  Bit 0: set 0 means input is signed message.
- #define HSM_OP_IMPORT_KEY_FLAGS_STRICT_OPERATION ((hsm_op_import_key_flags_t)(1u << 7))

  Bit 7: Strict: Request completed - New key written to NVM with updated MC.
- #define HSM_KEY_USAGE_EXPORT ((hsm_key_usage_t) (1u << 0))
- #define HSM_KEY_USAGE_ENCRYPT ((hsm_key_usage_t) (1u << 8))
- #define HSM_KEY_USAGE_DECRYPT ((hsm_key_usage_t) (1u << 9))
- #define HSM_KEY_USAGE_SIGN_MSG ((hsm_key_usage_t) (1u << 10))
- #define HSM_KEY_USAGE_VERIFY_MSG ((hsm_key_usage_t) (1u << 11))
- #define HSM_KEY_USAGE_SIGN_HASH ((hsm_key_usage_t) (1u << 12))
- #define HSM_KEY_USAGE_VERIFY_HASH ((hsm_key_usage_t) (1u << 13))
- #define HSM_KEY_USAGE_DERIVE ((hsm_key_usage_t) (1u << 14))
- #define HSM_KEY_INFO_PERSISTENT ((hsm_key_info_t)(0u << 1))
- #define HSM_KEY_INFO_PERMANENT ((hsm_key_info_t)(1u << 0))
- #define HSM_KEY_INFO_TRANSIENT ((hsm_key_info_t)(1u << 1))
- #define HSM_KEY_INFO_MASTER ((hsm_key_info_t)(1u << 2))
- #define HSM_KEY_INFO_KEK ((hsm_key_info_t)(1u << 3))
- #define HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION ((hsm_op_key_gen_flags_t)(1u << 7))
- #define HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN ((hsm_op_manage_key_group_flags_t)(1u << 0))
- #define HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_UNLOCK ((hsm_op_manage_key_group_flags_t)(1u << 1))

  Import the key group.
- #define HSM_OP_MANAGE_KEY_GROUP_FLAGS_IMPORT ((hsm_op_manage_key_group_flags_t)(1u << 2))

  Export the key group.
- #define HSM_OP_MANAGE_KEY_GROUP_FLAGS_EXPORT ((hsm_op_manage_key_group_flags_t)(1u << 3))
- #define HSM_OP_MANAGE_KEY_GROUP_FLAGS_MONOTONIC ((hsm_op_manage_key_group_flags_t)(1u << 5))
- #define HSM_OP_MANAGE_KEY_GROUP_FLAGS_SYNC_KEYSTORE ((hsm_op_manage_key_group_flags_t)(1u << 6))
- #define **HSM_OP_MANAGE_KEY_GROUP_FLAGS_STRICT_OPERATION** ((hsm_op_manage_key_group_flags_t)(1u << 7))

Typedefs

- typedef uint8_t hsm_op_delete_key_flags_t
- typedef uint8_t hsm_op_import_key_flags_t
- typedef uint32_t hsm_key_usage_t
- typedef uint16_t hsm_key_group_t
- typedef uint16_t hsm_key_info_t
- typedef uint8_t hsm_op_key_gen_flags_t

  Reserverd Bits 0 - 6.
- typedef uint8_t hsm_svc_key_management_flags_t
- typedef uint8_t **hsm_op_manage_key_group_flags_t**

Enumerations

- enum hsm_storage_loc_t { **HSM_SE_KEY_STORAGE** = 0x00000000 }
- enum hsm_storage_persist_lvl_t {
  HSM_VOLATILE_STORAGE = 0x0,
  HSM_PERSISTENT_STORAGE = 0x1,
  HSM_PERMANENT_STORAGE = 0xFF }
- enum hsm_key_lifetime_t {
  HSM_SE_KEY_STORAGE_VOLATILE = HSM_SE_KEY_STORAGE | HSM_VOLATILE_STORAGE,
  HSM_SE_KEY_STORAGE_PERSISTENT = HSM_SE_KEY_STORAGE | HSM_PERSISTENT_STORAGE,
  HSM_SE_KEY_STORAGE_PERS_PERM = HSM_SE_KEY_STORAGE | HSM_PERMANENT_STORAGE
  }
- enum hsm_pubkey_type_t {
  HSM_PUBKEY_TYPE_RSA = 0x4001,
  HSM_PUBKEY_TYPE_ECC_BP_R1 = 0x4130,
  HSM_PUBKEY_TYPE_ECC_NIST = 0x4112,
  HSM_PUBKEY_TYPE_ECC_BP_T1 = 0xC180 }
- enum hsm_key_type_t {
  HSM_KEY_TYPE_HMAC = 0x1100,
  HSM_KEY_TYPE_AES = 0x2400,
  HSM_KEY_TYPE_SM4 = 0x2405,
  HSM_KEY_TYPE_RSA = 0x7001,
  HSM_KEY_TYPE_ECC_BP_R1 = 0x7130,
  HSM_KEY_TYPE_ECC_NIST = 0x7112 }
- enum hsm_bit_key_sz_t {
  HSM_KEY_SIZE_HMAC_224 = 224,
  HSM_KEY_SIZE_HMAC_256 = 256,
  HSM_KEY_SIZE_HMAC_384 = 384,
  HSM_KEY_SIZE_HMAC_512 = 512,
  HSM_KEY_SIZE_AES_128 = 128,
  HSM_KEY_SIZE_AES_192 = 192,
  HSM_KEY_SIZE_AES_256 = 256,
  HSM_KEY_SIZE_SM4_128 = 128,
  HSM_KEY_SIZE_RSA_2048 = 2048,
  HSM_KEY_SIZE_RSA_3072 = 3072,
  HSM_KEY_SIZE_RSA_4096 = 4096,
  HSM_KEY_SIZE_ECC_BP_R1_224 = 224,
  HSM_KEY_SIZE_ECC_BP_R1_256 = 256,
  HSM_KEY_SIZE_ECC_BP_R1_320 = 320,
  HSM_KEY_SIZE_ECC_BP_R1_384 = 384,
  HSM_KEY_SIZE_ECC_BP_R1_512 = 512,
  HSM_KEY_SIZE_ECC_NIST_224 = 224,
  HSM_KEY_SIZE_ECC_NIST_256 = 256,
  HSM_KEY_SIZE_ECC_NIST_384 = 384,

HSM_KEY_SIZE_ECC_NIST_521 = 521,
HSM_KEY_SIZE_ECC_BP_T1_224 = 224,
HSM_KEY_SIZE_ECC_BP_T1_256 = 256,
HSM_KEY_SIZE_ECC_BP_T1_320 = 320,
HSM_KEY_SIZE_ECC_BP_T1_384 = 384 }

- enum hsm_permitted_algo_t {
PERMITTED_ALGO_SHA224 = ALGO_HASH_SHA224,
PERMITTED_ALGO_SHA256 = ALGO_HASH_SHA256,
PERMITTED_ALGO_SHA384 = ALGO_HASH_SHA384,
PERMITTED_ALGO_SHA512 = ALGO_HASH_SHA512,
PERMITTED_ALGO_SM3 = ALGO_HASH_SM3,
PERMITTED_ALGO_HMAC_SHA256 = ALGO_HMAC_SHA256,
PERMITTED_ALGO_HMAC_SHA384 = ALGO_HMAC_SHA384,
PERMITTED_ALGO_CMAC = ALGO_CMAC,
PERMITTED_ALGO_CTR = ALGO_CIPHER_CTR,
PERMITTED_ALGO_CFB = ALGO_CIPHER_CFB,
PERMITTED_ALGO_OFB = ALGO_CIPHER_OFB,
PERMITTED_ALGO_ECB_NO_PADDING = ALGO_CIPHER_ECB_NO_PAD,
PERMITTED_ALGO_CBC_NO_PADDING = ALGO_CIPHER_CBC_NO_PAD,
PERMITTED_ALGO_CCM = ALGO_CCM,
PERMITTED_ALGO_GCM = ALGO_GCM,
PERMITTED_ALGO_RSA_PKCS1_V15_SHA224 = ALGO_RSA_PKCS1_V15_SHA224,
PERMITTED_ALGO_RSA_PKCS1_V15_SHA256 = ALGO_RSA_PKCS1_V15_SHA256,
PERMITTED_ALGO_RSA_PKCS1_V15_SHA384 = ALGO_RSA_PKCS1_V15_SHA384,
PERMITTED_ALGO_RSA_PKCS1_V15_SHA512 = ALGO_RSA_PKCS1_V15_SHA512,
PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA224 = ALGO_RSA_PKCS1_PSS_MGF1_SHA224,
PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA256 = ALGO_RSA_PKCS1_PSS_MGF1_SHA256,
PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA384 = ALGO_RSA_PKCS1_PSS_MGF1_SHA384,
PERMITTED_ALGO_RSA_PKCS1_PSS_MGF1_SHA512 = ALGO_RSA_PKCS1_PSS_MGF1_SHA512,
PERMITTED_ALGO_ECDSA_SHA224 = ALGO_ECDSA_SHA224,
PERMITTED_ALGO_ECDSA_SHA256 = ALGO_ECDSA_SHA256,
PERMITTED_ALGO_ECDSA_SHA384 = ALGO_ECDSA_SHA384,
PERMITTED_ALGO_ECDSA_SHA512 = ALGO_ECDSA_SHA512,
PERMITTED_ALGO_HMAC_KDF_SHA256 = ALGO_HMAC_KDF_SHA256,
PERMITTED_ALGO_ALL_CIPHER = ALGO_CIPHER_ALL,
PERMITTED_ALGO_ALL_AEAD = ALGO_ALL_AEAD,
PERMITTED_ALGO_OTH_KEK_CBC = ALGO_CIPHER_KEK_CBC }
- enum hsm_key_lifecycle_t {
HSM_KEY_LIFECYCLE_OPEN = 0x1,
HSM_KEY_LIFECYCLE_CLOSED = 0x2,
HSM_KEY_LIFECYCLE_CLOSED_LOCKED = 0x4 }

Functions

- hsm_err_t hsm_delete_key (hsm_hdl_t key_management_hdl, op_delete_key_args_t *args)
- hsm_err_t hsm_get_key_attr (hsm_hdl_t key_management_hdl, op_get_key_attr_args_t *args)
- hsm_err_t hsm_import_key (hsm_hdl_t key_management_hdl, op_import_key_args_t *args)
- hsm_err_t hsm_generate_key_ext (hsm_hdl_t key_management_hdl, op_generate_key_ext_args_t *args)
- hsm_err_t hsm_generate_key (hsm_hdl_t key_management_hdl, op_generate_key_args_t *args)
- hsm_err_t hsm_open_key_management_service (hsm_hdl_t key_store_hdl, open_svc_key_management_args_t *args, hsm_hdl_t *key_management_hdl)
- hsm_err_t hsm_close_key_management_service (hsm_hdl_t key_management_hdl)
- hsm_err_t hsm_manage_key_group (hsm_hdl_t key_management_hdl, op_manage_key_group_args_t *args)

  The entire key group will be cached in the HSM local memory.

5.2.1 Detailed Description

5.2.2 Data Structure Documentation

5.2.2.1 struct op_delete_key_args_t Structure detailing the delete key operation member arguments

Data Fields

uint32_t	key_identifier	identifier of the key to be used for the operation.
hsm_op_delete_key_flags_t	flags	bitmap specifying the operation properties.

5.2.2.2 struct op_get_key_attr_args_t Structure describing the get key attribute operation arguments

Data Fields

uint32_t	key_identifier	identifier of the key to be used for the operation.
hsm_key_type_t	key_type	indicates which type of key must be generated.
hsm_bit_key_sz_t	bit_key_sz	indicates key security size in bits.
hsm_key_lifetime_t	key_lifetime	this attribute comprises of two indicaters-key persistence level and location where the key is stored.
hsm_key_usage_t	key_usage	indicates the cryptographic operations that key can execute.
hsm_permitted_algo_t	permitted_algo	indicates the key permitted algorithm.
hsm_key_lifecycle_t	lifecycle	indicates the device lifecycle in which key is usable.

5.2.2.3 struct op_import_key_args_t Structure detailing the import key operation member arguments

Data Fields

uint32_t	key_identifier	Identifier of the KEK used to encrypt the key to be imported (Ignored if KEK is not used as set as part of "flags" field).
uint8_t *	input_lsb_addr	Address in the requester space where: • EdgeLock 2GO TLV can be found. • Ignore this field if not E2GO_TLV.
uint32_t	input_size	Size in bytes of: • EdgeLock 2GO TLV can be found. • Ignore this field if not E2GO_TLV.
hsm_op_import_key_flags_t	flags	bitmap specifying the operation properties.

5.2.2.4 struct kek_enc_key_hdr_t Structure describing the encryption key header

Data Fields

uint8_t	iv[IV_LENGTH]	

Data Fields

uint8_t *	key	
uint32_t	tag	

5.2.2.5 struct op_generate_key_ext_args_t Structure detailing the key generate operation member arguments

Data Fields

uint32_t *	key_identifier	pointer to the identifier of the key to be used for the operation In case of create operation the new key identifier will be stored in this location
uint16_t	out_size	length in bytes of the generated key It must be 0 in case of symmetric keys
hsm_op_key_gen_flags_t	flags	bitmap specifying the operation properties
hsm_key_type_t	key_type	indicates which type of key must be generated
hsm_key_group_t	key_group	Key group of the generated key. It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the hsm_manage_key_group API
hsm_key_info_t	key_info	bitmap specifying the properties of the key
uint8_t *	out_key	pointer to the output area where the generated public key must be written.
uint8_t	min_mac_len	min mac length in bits to be set for this key, value 0 indicates use default (see op_mac_one_go_args_t for more details). Only accepted for keys that can be used for mac operations, must not be larger than maximum mac size that can be performed with the key. When in FIPS approved mode values $<$ 32 bits are not allowed.
uint8_t	reserved[3]	It must be 0.

5.2.2.6 struct op_generate_key_args_t Structure describing the generate key operation member arguments

Data Fields

uint32_t *	key_identifier	pointer to the identifier of the key to be used for the operation. In case of create operation the new key identifier will be stored in this location.
uint16_t	out_size	length in bytes of the generated key.It must be 0 in case of symmetric keys.
hsm_op_key_gen_flags_t	flags	bitmap specifying the operation properties.
hsm_key_type_t	key_type	indicates which type of key must be generated.
hsm_key_group_t	key_group	Key group of the generated key. It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the hsm_manage_key_group API.
uint8_t *	out_key	pointer to the output area where the generated public key must be written.
uint16_t	exp_out_size	expected output key buffer size, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code
hsm_bit_key_sz_t	bit_key_sz	indicates key security size in bits.
hsm_key_lifecycle_t	key_lifecycle	defines the key lifecycle in which the key is usable. If it is set to 0, current key lifecycle is used.

Data Fields

hsm_key_lifetime_t	key_lifetime	this attribute comprises of two indicaters-key persistence level and location where the key is stored.
hsm_key_usage_t	key_usage	indicates the cryptographic operations that key can execute.
hsm_permitted_algo_t	permitted_algo	indicates the key permitted algorithm.

5.2.2.7 struct open_svc_key_management_args_t

Structure detailing the key management open service member arguments

Data Fields

hsm_hdl_t	key_management_hdl	handle identifying the key management service flow
hsm_svc_key_management_flags_t	flags	bitmap specifying the services properties.

Data Fields

hsm_key_group_t	key_group	It must be a value in the range 0-1023. Keys belonging to the same group can be cached in the HSM local memory through the hsm_manage_key_group API.
hsm_op_manage_key_group_flags_t	flags	bitmap specifying the operation properties.
uint8_t	reserved	

5.2.2.8 struct op_manage_key_group_args_t

5.2.3 Macro Definition Documentation

5.2.3.1 HSM_OP_DEL_KEY_FLAGS_STRICT_OPERATION

```
#define HSM_OP_DEL_KEY_FLAGS_STRICT_OP↩
ERATION ((hsm_op_import_key_flags_t)(1u << 7))
```

Bitmap detailing the delete key operation properties. Bit 0-6: Reserved. Bit 7: Strict: Request completed - New key written to NVM with updated MC.

5.2.3.2 HSM_KEY_USAGE_EXPORT

```
#define HSM_KEY_USAGE_EXPORT ((hsm_key_usage_t) (1u << 0))
```

Bit indicating the permission to export the key

5.2.3.3 HSM_KEY_USAGE_ENCRYPT

```
#define HSM_KEY_USAGE_ENCRYPT ((hsm_key_usage_t) (1u <<
8))
```

Bit indicating the permission to encrypt a message with the key

5.2.3.4 HSM_KEY_USAGE_DECRYPT

```
#define HSM_KEY_USAGE_DECRYPT ((hsm_key_usage_t) (1u << 9))
```

Bit indicating the permission to decrypt a message with the key

5.2.3.5 HSM_KEY_USAGE_SIGN_MSG

```
#define HSM_KEY_USAGE_SIGN_MSG ((hsm_key_usage_t) (1u << 10))
```

Bit indicating the permission to sign a message with the key

5.2.3.6 HSM_KEY_USAGE_VERIFY_MSG

```
#define HSM_KEY_USAGE_VERIFY_MSG ((hsm_key_usage_t) (1u << 11))
```

Bit indicating the permission to verify a message signature with the key

5.2.3.7 HSM_KEY_USAGE_SIGN_HASH

```
#define HSM_KEY_USAGE_SIGN_HASH ((hsm_key_usage_t) (1u << 12))
```

Bit indicating the permission to sign a hashed message with the key

5.2.3.8 HSM_KEY_USAGE_VERIFY_HASH

```
#define HSM_KEY_USAGE_VERIFY_HASH ((hsm_key_usage_t) (1u << 13))
```

Bit indicating the permission to verify a hashed message signature with the key

5.2.3.9 HSM_KEY_USAGE_DERIVE

```
#define HSM_KEY_USAGE_DERIVE ((hsm_key_usage_t) (1u << 14))
```

Bit indicating the permission to derive other keys from this key

5.2.3.10 HSM_KEY_INFO_PERSISTENT

```
#define HSM_KEY_INFO_PERSISTENT ((hsm_key_info_t)(0u << 1))
```

Bit indicating persistent keys which are stored in the external NVM. The entire key group is written in the NVM at the next STRICT operation.

5.2.3.11 HSM_KEY_INFO_PERMANENT

```
#define HSM_KEY_INFO_PERMANENT ((hsm_key_info_t)(1u << 0))
```

Bit indicating the key is permanent. When set, the key is permanent (write locked). Once created, it will not be possible to update or delete the key anymore. Transient keys will be anyway deleted after a PoR or when the corresponding key store service flow is closed. This bit can never be reset.

5.2.3.12 HSM_KEY_INFO_TRANSIENT

```
#define HSM_KEY_INFO_TRANSIENT ((hsm_key_info_t)(1u << 1))
```

Bit indicating the key is transient. Transient keys are deleted when the corresponding key store service flow is closed or after a PoR. Transient keys cannot be in the same key group than persistent keys.

5.2.3.13 HSM_KEY_INFO_MASTER `#define HSM_KEY_INFO_MASTER ((hsm_key_info_t)(1u << 2))`

Bit indicating the key is master key. When set, the key is considered as a master key. Only master keys can be used as input of key derivation functions (i.e butterfly key expansion).

5.2.3.14 HSM_KEY_INFO_KEK `#define HSM_KEY_INFO_KEK ((hsm_key_info_t)(1u << 3))`

Bit indicating the key is key encryption key When set, the key is considered as a key encryption key. KEK keys can only be used to wrap and import other keys into the key store, all other operation are not allowed. Only keys imported in the key store through the hsm_mange_key API can get this attribute.

5.2.3.15 HSM_OP_KEY_GENERATION_FLAGS_STRICT_OPERATION `#define HSM_OP_KEY_GENERATION↩_FLAGS_STRICT_OPERATION ((hsm_op_key_gen_flags_t)(1u << 7))`

The request is completed only when the new key has been written in the NVM. This applicable for persistent key only.

5.2.3.16 HSM_OP_MANAGE_KEY_GROUP_FLAGS_CACHE_LOCKDOWN `#define HSM_OP_MANAGE_KEY↩_GROUP_FLAGS_CACHE_LOCKDOWN ((hsm_op_manage_key_group_flags_t)(1u << 0))`

HSM may export the key group in the external NVM to free up the local memory. HSM will copy the key group in the local memory again in case of key group usage/update.

5.2.3.17 HSM_OP_MANAGE_KEY_GROUP_FLAGS_EXPORT `#define HSM_OP_MANAGE_KEY_GROUP_FLAG↩S_EXPORT ((hsm_op_manage_key_group_flags_t)(1u << 3))`

When used in conjunction with SYNC key group or SYNC key store and storage only, the request is completed only when the monotonic counter has been updated.

5.2.3.18 HSM_OP_MANAGE_KEY_GROUP_FLAGS_MONOTONIC `#define HSM_OP_MANAGE_KEY_GROUP_↩FLAGS_MONOTONIC ((hsm_op_manage_key_group_flags_t)(1u << 5))`

The request is completed only when the update has been written in the NVM. Not applicable for cache lock-down/unlock.

5.2.3.19 HSM_OP_MANAGE_KEY_GROUP_FLAGS_SYNC_KEYSTORE `#define HSM_OP_MANAGE_KEY_G↩ROUP_FLAGS_SYNC_KEYSTORE ((hsm_op_manage_key_group_flags_t)(1u << 6))`

The request is completed only when the update has been written in the NVM. Not applicable for cache lock-down/unlock.

5.2.4 Typedef Documentation

5.2.4.1 hsm_op_delete_key_flags_t `typedef uint8_t hsm_op_delete_key_flags_t`

Bitmap describing the delete key operation properties

5.2.4.2 hsm_op_import_key_flags_t typedef uint8_t hsm_op_import_key_flags_t

Bitmap specifying the import key operation supported properties Bit 0: Defines input configuration Bit 1-6: Reserved Bit 7: Strict

5.2.4.3 hsm_key_usage_t typedef uint32_t hsm_key_usage_t

Bitmap indicating the cryptographic operations that key can execute

5.2.4.4 hsm_key_group_t typedef uint16_t hsm_key_group_t

Bit field indicating the key group

5.2.4.5 hsm_key_info_t typedef uint16_t hsm_key_info_t

Bit field indicating the key information

5.2.4.6 hsm_op_key_gen_flags_t typedef uint8_t hsm_op_key_gen_flags_t

Reserverd Bits 0 - 6.

Bitmap specifying the key generate operation supported properties.

5.2.4.7 hsm_svc_key_management_flags_t typedef uint8_t hsm_svc_key_management_flags_t

Bitmap specifying the key management service supported properties

5.2.5 Enumeration Type Documentation

5.2.5.1 hsm_storage_loc_t enum hsm_storage_loc_t

Enum Indicating the key location indicator.

5.2.5.2 hsm_storage_persist_lvl_t enum hsm_storage_persist_lvl_t

Enum Indicating the key persistent level indicator.

5.2.5.3 hsm_key_lifetime_t enum hsm_key_lifetime_t

Enum Indicating the key lifetime.

5.2.5.4 hsm_pubkey_type_t enum hsm_pubkey_type_t

Enum Indicating the public key type.

5.2.5.5 hsm_key_type_t enum hsm_key_type_t

Enum Indicating the key type.

5.2.5.6 hsm_bit_key_sz_t enum hsm_bit_key_sz_t

Enum Indicating the key security size in bits.

5.2.5.7 hsm_permitted_algo_t enum hsm_permitted_algo_t

Enum describing the permiteed algorithm

5.2.5.8 hsm_key_lifecycle_t enum hsm_key_lifecycle_t

Enum detailing Permitted key lifecycle

5.2.6 Function Documentation

5.2.6.1 hsm_delete_key()

```
hsm_err_t hsm_delete_key (
        hsm_hdl_t key_management_hdl,
        op_delete_key_args_t * args )
```

This command is designed to perform the following operations:

- delete an existing key

Parameters

key_management_hdl	handle identifying the key management service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.2.6.2 hsm_get_key_attr()

```
hsm_err_t hsm_get_key_attr (
        hsm_hdl_t key_management_hdl,
        op_get_key_attr_args_t * args )
```

This command is designed to perform the following operations:

- get attributes of an existing key

Parameters

key_management_hdl	handle identifying the key management service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.2.6.3 hsm_import_key()

```
hsm_err_t hsm_import_key (
        hsm_hdl_t key_management_hdl,
        op_import_key_args_t * args )
```

This API will be used to Import the key

Parameters

key_management_hdl	handle identifying the key management service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.2.6.4 hsm_generate_key_ext()

```
hsm_err_t hsm_generate_key_ext (
        hsm_hdl_t key_management_hdl,
        op_generate_key_ext_args_t * args )
```

Generate a key or a key pair with extended settings. Basic operation is identical to hsm_generate_key, but accepts additional settings. Currently the min_mac_len is the only additional setting accepted.

Parameters

key_management_hdl	handle identifying the key management service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.2.6.5 hsm_generate_key()

```
hsm_err_t hsm_generate_key (
        hsm_hdl_t key_management_hdl,
        op_generate_key_args_t * args )
```

Generate a key or a key pair. Only the confidential keys (symmetric and private keys) are stored in the internal key store, while the non-confidential keys (public key) are exported.

The generated key can be stored using a new or existing key identifier with the restriction that an existing key can be replaced only by a key of the same type.

Parameters

key_management_hdl	handle identifying the key management service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.2.6.6 hsm_open_key_management_service()

```
hsm_err_t hsm_open_key_management_service (
        hsm_hdl_t key_store_hdl,
        open_svc_key_management_args_t * args,
        hsm_hdl_t * key_management_hdl )
```

Open a key management service flow
User must open this service flow in order to perform operation on the key store keys (generate, update, delete)

Parameters

key_store_hdl	handle identifying the key store service flow.
args	pointer to the structure containing the function arguments.
key_management_hdl	pointer to where the key management service flow handle must be written.

Returns

error code.

5.2.6.7 hsm_close_key_management_service()

```
hsm_err_t hsm_close_key_management_service (
        hsm_hdl_t key_management_hdl )
```

Terminate a previously opened key management service flow

Parameters

key_management_hdl	handle identifying the key management service flow.

Returns

error code

5.2.6.8 hsm_manage_key_group()

```
hsm_err_t hsm_manage_key_group (
        hsm_hdl_t key_management_hdl,
        op_manage_key_group_args_t * args )
```

The entire key group will be cached in the HSM local memory.

This command is designed to perform the following operations:

- lock/unlock down a key group in the HSM local memory so that the keys are available to the HSM without additional latency
- un-lock a key group. HSM may export the key group into the external NVM to free up local memory as needed
- delete an existing key group

User can call this function only after having opened a key management service flow.

Parameters

key_management_hdl	handle identifying the key management service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.3 Ciphering

Data Structures

- struct op_auth_enc_args_t
- struct open_svc_cipher_args_t
- struct op_cipher_one_go_args_t

Macros

- #define HSM_AUTH_ENC_FLAGS_DECRYPT ((hsm_op_auth_enc_flags_t)(0u << 0))
- #define HSM_AUTH_ENC_FLAGS_ENCRYPT ((hsm_op_auth_enc_flags_t)(1u << 0))
- #define HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV ((hsm_op_auth_enc_flags_t)(1u << 1))
- #define HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV ((hsm_op_auth_enc_flags_t)(1u << 2))
- #define HSM_CIPHER_ONE_GO_FLAGS_DECRYPT ((hsm_op_cipher_one_go_flags_t)(0u << 0))
- #define HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT ((hsm_op_cipher_one_go_flags_t)(1u << 0))

Typedefs

- typedef uint8_t hsm_op_auth_enc_flags_t
- typedef uint8_t hsm_svc_cipher_flags_t
- typedef uint8_t hsm_op_cipher_one_go_flags_t

Enumerations

- enum hsm_op_auth_enc_algo_t { HSM_AEAD_ALGO_CCM = ALGO_CCM }
- enum hsm_op_cipher_one_go_algo_t {
  HSM_CIPHER_ONE_GO_ALGO_CTR = ALGO_CIPHER_CTR,
  HSM_CIPHER_ONE_GO_ALGO_CFB = ALGO_CIPHER_CFB,
  HSM_CIPHER_ONE_GO_ALGO_OFB = ALGO_CIPHER_OFB,
  HSM_CIPHER_ONE_GO_ALGO_ECB = ALGO_CIPHER_ECB_NO_PAD,
  HSM_CIPHER_ONE_GO_ALGO_CBC = ALGO_CIPHER_CBC_NO_PAD }

Functions

- hsm_err_t hsm_do_cipher (hsm_hdl_t cipher_hdl, op_cipher_one_go_args_t *cipher_one_go)
- hsm_err_t hsm_auth_enc (hsm_hdl_t cipher_hdl, op_auth_enc_args_t *args)
- hsm_err_t hsm_open_cipher_service (hsm_hdl_t key_store_hdl, open_svc_cipher_args_t *args, hsm_hdl_t *cipher_hdl)
- hsm_err_t hsm_cipher_one_go (hsm_hdl_t cipher_hdl, op_cipher_one_go_args_t *args)
- hsm_err_t hsm_close_cipher_service (hsm_hdl_t cipher_hdl)

5.3.1 Detailed Description

5.3.2 Data Structure Documentation

5.3.2.1 struct op_auth_enc_args_t Structure describing the authenticated encryption operation arguments

Data Fields

uint32_t	key_identifier	identifier of the key to be used for the operation
uint8_t *	iv	pointer to the user supplied part of initialization vector or nonce, when applicable, otherwise 0
uint16_t	iv_size	length in bytes of the fixed part of the initialization vector for encryption (0 or 4 bytes), length in bytes of the full IV for decryption (12 bytes)
uint8_t *	aad	pointer to the additional authentication data
uint16_t	aad_size	length in bytes of the additional authentication data
hsm_op_auth_enc_algo_t	ae_algo	algorithm to be used for the operation
hsm_op_auth_enc_flags_t	flags	bitmap specifying the operation attributes
uint8_t *	input	pointer to the input area plaintext for encryption Ciphertext + Tag (16 bytes) for decryption
uint8_t *	output	pointer to the output area Ciphertext + Tag (16 bytes) • IV for encryption plaintext for decryption if the Tag is verified
uint32_t	input_size	length in bytes of the input
uint32_t	output_size	length in bytes of the output
uint32_t	exp_output_size	expected output buffer size in bytes, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code

5.3.2.2 struct open_svc_cipher_args_t Structure describing the open cipher service members

Data Fields

uint32_t	cipher_hdl	handle identifying the cipher service flow
uint8_t	flags	bitmap specifying the services properties
uint8_t	reserved[3]	

5.3.2.3 struct op_cipher_one_go_args_t Structure describing the cipher one go operation arguments

Data Fields

uint32_t	key_identifier	identifier of the key to be used for the operation
uint8_t *	iv	pointer to the initialization vector (nonce in case of AES CCM)
uint16_t	iv_size	length in bytes of the initialization vector. it must be 0 for algorithms not using the initialization vector. It must be 12 for AES in CCM mode
uint8_t	svc_flags	bitmap specifying the services properties.
uint8_t	flags	bitmap specifying the operation attributes
uint32_t	cipher_algo	algorithm to be used for the operation
uint8_t *	input	pointer to the input area: • plaintext for encryption • ciphertext for decryption Note: In case of CCM it is the purported ciphertext.

Data Fields

uint8_t *	output	pointer to the output area: • ciphertext for encryption Note: In case of CCM it is the output of the generation-encryption process. • plaintext for decryption
uint32_t	input_size	length in bytes of the input. • In case of CBC and ECB, the input size should be multiple of a block cipher size (16 bytes).
uint32_t	output_size	length in bytes of the output
uint32_t	exp_output_size	expected output buffer size in bytes, valid in case of (0x1D) error code

5.3.3 Macro Definition Documentation

5.3.3.1 HSM_AUTH_ENC_FLAGS_DECRYPT `#define HSM_AUTH_ENC_FLAGS_DECRYPT ((hsm_op_auth_enc_flags_t)(0u << 0))`

Bit indicating the decryption operation

5.3.3.2 HSM_AUTH_ENC_FLAGS_ENCRYPT `#define HSM_AUTH_ENC_FLAGS_ENCRYPT ((hsm_op_auth_enc_flags_t)(1u << 0))`

Bit indicating the encryption operation

5.3.3.3 HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV `#define HSM_AUTH_ENC_FLAGS_GENERATE_FULL_↩ IV ((hsm_op_auth_enc_flags_t)(1u << 1))`

Bit indicating the Full IV is internally generated (only relevant for encryption)

5.3.3.4 HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV `#define HSM_AUTH_ENC_FLAGS_GENERATE_↩ COUNTER_IV ((hsm_op_auth_enc_flags_t)(1u << 2))`

Bit indicating 4 bytes supplied other bytes internally generated (only relevant for encryption)

5.3.3.5 HSM_CIPHER_ONE_GO_FLAGS_DECRYPT `#define HSM_CIPHER_ONE_GO_FLAGS_DECRYPT ((hsm_op_cipher_one_go << 0))`

Bit indicating the decrypt operation

5.3.3.6 HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT `#define HSM_CIPHER_ONE_GO_FLAGS_ENCRYPT ((hsm_op_cipher_one_go << 0))`

Bit indicating the encrypt operation

5.3.4 Typedef Documentation

5.3.4.1 hsm_op_auth_enc_flags_t `typedef uint8_t hsm_op_auth_enc_flags_t`

Bit field indicating the authenticated encryption operations

5.3.4.2 hsm_svc_cipher_flags_t `typedef uint8_t hsm_svc_cipher_flags_t`

Bit field describing the open cipher service requested operation

5.3.4.3 hsm_op_cipher_one_go_flags_t `typedef uint8_t hsm_op_cipher_one_go_flags_t`

Bit field indicating the requested operations

5.3.5 Enumeration Type Documentation

5.3.5.1 hsm_op_auth_enc_algo_t `enum hsm_op_auth_enc_algo_t`

Bit field indicating the supported algorithm

Enumerator

HSM_AEAD_ALGO_CCM	CCM (AES CCM)

5.3.5.2 hsm_op_cipher_one_go_algo_t `enum hsm_op_cipher_one_go_algo_t`

Enum describing the cipher one go operation algorithm

Enumerator

HSM_CIPHER_ONE_GO_ALGO_CTR	CTR (AES supported). CFB (AES supported).
HSM_CIPHER_ONE_GO_ALGO_CFB	OFB (AES supported).
HSM_CIPHER_ONE_GO_ALGO_OFB	ECB no padding (AES, SM4 supported).
HSM_CIPHER_ONE_GO_ALGO_ECB	CBC no padding (AES, SM4 supported).

5.3.6 Function Documentation

5.3.6.1 hsm_do_cipher()

```
hsm_err_t hsm_do_cipher (
        hsm_hdl_t cipher_hdl,
        op_cipher_one_go_args_t * cipher_one_go )
```

Secondary API to perform ciphering operation
This API does the following:

1. Open an Cipher Service Flow
2. Perform ciphering operation
3. Terminate a previously opened cipher service flow
   User can call this function only after having opened a cipher service flow.

Parameters

cipher_hdl	handle identifying the cipher service flow.
cipher_one_go	pointer to the structure containing the function arguments.

Returns

error code

5.3.6.2 hsm_auth_enc()

```
hsm_err_t hsm_auth_enc (
        hsm_hdl_t cipher_hdl,
        op_auth_enc_args_t * args )
```

Perform authenticated encryption operation
User can call this function only after having opened a cipher service flow
For decryption operations, the full IV is supplied by the caller via the iv and iv_size parameters. HSM_AUTH_EN↩C_FLAGS_GENERATE_FULL_IV and HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV flags are ignored.
For encryption operations, either HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV or HSM_AUTH_ENC_FLA↩GS_GENERATE_COUNTER_IV must be set when calling this function:

- When HSM_AUTH_ENC_FLAGS_GENERATE_FULL_IV is set, the full IV is internally generated, iv and iv_size must be set to 0
- When HSM_AUTH_ENC_FLAGS_GENERATE_COUNTER_IV is set, the user supplies a 4 byte fixed part of the IV. The other IV bytes are internally generated

Parameters

cipher_hdl	handle identifying the cipher service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.3.6.3 hsm_open_cipher_service()

```
hsm_err_t hsm_open_cipher_service (
        hsm_hdl_t key_store_hdl,
        open_svc_cipher_args_t * args,
        hsm_hdl_t * cipher_hdl )
```

- Open a cipher service flow.
- User can call this function only after having opened a key-store service flow.
- User must open this service in order to perform cipher operation.

Parameters

key_store_hdl	handle identifying the key store service flow.
args	pointer to the structure containing the function arguments.
cipher_hdl	pointer to where the cipher service flow handle must be written.

Returns

error code

5.3.6.4 hsm_cipher_one_go()

```
hsm_err_t hsm_cipher_one_go (
        hsm_hdl_t cipher_hdl,
        op_cipher_one_go_args_t * args )
```

Perform ciphering operation
User can call this function only after having opened a cipher service flow

Parameters

cipher_hdl	handle identifying the cipher service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.3.6.5 hsm_close_cipher_service()

```
hsm_err_t hsm_close_cipher_service (
        hsm_hdl_t cipher_hdl )
```

Terminate a previously opened cipher service flow

Parameters

cipher_hdl	pointer to handle identifying the cipher service flow to be closed.

Returns

error code

5.4 Signature generation

Data Structures

- struct open_svc_sign_gen_args_t
- struct op_generate_sign_args_t
- struct op_prepare_sign_args_t

Macros

- #define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_generate_sign_flags_t)(0u << 0))
- #define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_generate_sign_flags_t)(1u << 0))
- #define HSM_OP_PREPARE_SIGN_INPUT_DIGEST ((hsm_op_prepare_signature_flags_t)(0u << 0))

  Bit indicating input digest.
- #define HSM_OP_PREPARE_SIGN_INPUT_MESSAGE ((hsm_op_prepare_signature_flags_t)(1u << 0))

  Bit indicating input message.
- #define HSM_OP_PREPARE_SIGN_COMPRESSED_POINT ((hsm_op_prepare_signature_flags_t)(1u << 1))

  Bit indicating compressed point.

Typedefs

- typedef uint8_t hsm_op_generate_sign_flags_t
- typedef uint8_t hsm_op_prepare_signature_flags_t

Enumerations

- enum hsm_signature_scheme_id_t {
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA224 = 0x06000208,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA256 = 0x06000209,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA384 = 0x0600020A,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_SHA512 = 0x0600020B,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_V15_ANY_HASH = 0x060002FF,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA224 = 0x06000308,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA256 = 0x06000309,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA384 = 0x0600030A,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_SHA512 = 0x0600030B,
  HSM_SIGNATURE_SCHEME_RSA_PKCS1_PSS_MGF1_ANY_HASH = 0x060003FF,
  HSM_SIGNATURE_SCHEME_ECDSA_ANY = 0x06000600,
  HSM_SIGNATURE_SCHEME_ECDSA_SHA224 = 0x06000608,
  HSM_SIGNATURE_SCHEME_ECDSA_SHA256 = 0x06000609,
  HSM_SIGNATURE_SCHEME_ECDSA_SHA384 = 0x0600060A,
  HSM_SIGNATURE_SCHEME_ECDSA_SHA512 = 0x0600060B }

Functions

- hsm_err_t hsm_do_sign (hsm_hdl_t key_store_hdl, op_generate_sign_args_t *args)
- hsm_err_t hsm_open_signature_generation_service (hsm_hdl_t key_store_hdl, open_svc_sign_gen_args_t *args, hsm_hdl_t *signature_gen_hdl)
- hsm_err_t hsm_close_signature_generation_service (hsm_hdl_t signature_gen_hdl)
- hsm_err_t hsm_generate_signature (hsm_hdl_t signature_gen_hdl, op_generate_sign_args_t *args)
- hsm_err_t hsm_prepare_signature (hsm_hdl_t signature_gen_hdl, op_prepare_sign_args_t *args)

5.4.1 Detailed Description

5.4.2 Data Structure Documentation

5.4.2.1 struct open_svc_sign_gen_args_t Structure to represent the generate sign open service arguments

Data Fields

hsm_hdl_t	signature_gen_hdl	

5.4.2.2 struct op_generate_sign_args_t Structure to represent the generate sign operation arguments

Data Fields

uint32_t	key_identifier	identifier of the key to be used for the operation
uint8_t *	message	pointer to the input (message or message digest) to be signed
uint8_t *	signature	pointer to the output area where the signature must be stored. The signature S=(r,s) is stored in format r||s||Ry where: • Ry is an additional byte containing the lsb of y. Ry has to be considered valid only if the HSM_OP_GENERATE_SIGN_FLAGS_CO↩ MPRESSED_POINT is set.
uint16_t	signature_size	length in bytes of the output. After signature generation operation, this field will contain the expected signature buffer size, if operation failed due to provided output buffer size being too short.
uint32_t	message_size	length in bytes of the input
hsm_signature_scheme_id_t	scheme_id	identifier of the digital signature scheme to be used for the operation
uint16_t	salt_len	Salt length in bytes.
uint16_t	exp_signature_size	expected signature buffer size for output, returned by FW in case the input signature size provided is less than the required size.
hsm_op_generate_sign_flags_t	flags	bitmap specifying the operation attributes

5.4.2.3 struct op_prepare_sign_args_t Structure detailing the prepare signature operation member arguments

Data Fields

hsm_signature_scheme_id_t	scheme_id	identifier of the digital signature scheme to be used for the operation.
hsm_op_prepare_signature_flags_t	flags	bitmap specifying the operation attributes

5.4.3 Macro Definition Documentation

5.4.3.1 HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST

```
#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_generate_sign_flags_t)(0u << 0))
```

Bit field indicating the input is the message digest

5.4.3.2 HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE

```
#define HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_generate_sign_flags_t)(1u << 0))
```

Bit field indicating the input is the actual message

5.4.4 Typedef Documentation

5.4.4.1 hsm_op_generate_sign_flags_t

```
typedef uint8_t hsm_op_generate_sign_flags_t
```

Bit field indicating the requested operation

5.4.4.2 hsm_op_prepare_signature_flags_t

```
typedef uint8_t hsm_op_prepare_signature_flags_t
```

Bitmap specifying the prepare signature operation supported attributes

5.4.5 Enumeration Type Documentation

5.4.5.1 hsm_signature_scheme_id_t

```
enum hsm_signature_scheme_id_t
```

Bit field indicating the PSA compliant requested operations: Bit 2 to 7: Reserved.

5.4.6 Function Documentation

5.4.6.1 hsm_do_sign()

```
hsm_err_t hsm_do_sign (
        hsm_hdl_t key_store_hdl,
        op_generate_sign_args_t * args )
```

Secondary API to generate signature on the given message.
This API does the following:

1. Open a service flow for signature generation.
2. Based on the flag to identify the type of message: Digest or actual message, generate the signature using the key corresponding to the key id.
3. Post performing the operation, terminate the previously opened signature-generation service flow.
   User can call this function only after having opened a key-store.

Parameters

key_store_hdl	handle identifying the current key-store.
args	pointer to the structure containing the function arguments.

Returns

error code

5.4.6.2 hsm_open_signature_generation_service()

```
hsm_err_t hsm_open_signature_generation_service
(
        hsm_hdl_t key_store_hdl,
        open_svc_sign_gen_args_t * args,
        hsm_hdl_t * signature_gen_hdl )
```

Open a signature generation service flow
User can call this function only after having opened a key store service flow.

User must open this service in order to perform signature generation operations.

Parameters

key_store_hdl	handle identifying the key store service flow.
args	pointer to the structure containing the function arguments.
signature_gen_hdl	pointer to where the signature generation service flow handle must be written.

Returns

error code

5.4.6.3 hsm_close_signature_generation_service()

```
hsm_err_t hsm_close_signature_generation_service (
        hsm_hdl_t signature_gen_hdl )
```

Terminate a previously opened signature generation service flow

Parameters

signature_gen_hdl	handle identifying the signature generation service flow to be closed.

Returns

error code

5.4.6.4 hsm_generate_signature() hsm_err_t hsm_generate_signature (

```
hsm_hdl_t signature_gen_hdl,
op_generate_sign_args_t * args )
```

Generate a digital signature according to the signature scheme User can call this function only after having opened a signature generation service flow.

The signature S=(r,s) is stored in the format r||s||Ry where:

- Ry is an additional byte containing the lsb of y. Ry has to be considered valid only if the HSM_OP_GENE↩ RATE_SIGN_FLAGS_COMPRESSED_POINT is set.

In case of HSM_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3, message of op_generate_sign_args_t should be (as specified in GB/T 32918):

- equal to Z||M in case of HSM_OP_GENERATE_SIGN_FLAGS_INPUT_MESSAGE
- equal to SM3(Z||M) in case of HSM_OP_GENERATE_SIGN_FLAGS_INPUT_DIGEST

Parameters

signature_gen_hdl	handle identifying the signature generation service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.4.6.5 hsm_prepare_signature() hsm_err_t hsm_prepare_signature (

```
hsm_hdl_t signature_gen_hdl,
op_prepare_sign_args_t * args )
```

Prepare the creation of a signature by pre-calculating the operations having not dependencies on the input message.

The pre-calculated value will be stored internally and used once call hsm_generate_signature. Up to 20 pre-calculated values can be stored, additional preparation operations will have no effects.

User can call this function only after having opened a signature generation service flow.

The signature S=(r,s) is stored in the format r||s||Ry where:

- Ry is an additional byte containing the lsb of y, Ry has to be considered valid only if the HSM_OP_PREPA↩ RE_SIGN_COMPRESSED_POINT is set.

Parameters

signature_gen_hdl	handle identifying the signature generation service flow
args	pointer to the structure containing the function arguments.

Returns

error code

5.5 Signature verification

Data Structures

- struct open_svc_sign_ver_args_t
- struct op_verify_sign_args_t

Macros

- #define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST ((hsm_op_verify_sign_flags_t)(0u << 0))
- #define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE ((hsm_op_verify_sign_flags_t)(1u << 0))
- #define HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT ((hsm_op_verify_sign_flags_t)(1u << 1))
- #define HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL ((hsm_op_verify_sign_flags_t)(1u << 2))
- #define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_status_t)(0x5A3CC3A5u))
- #define HSM_VERIFICATION_STATUS_FAILURE ((hsm_verification_status_t)(0x2B4DD4B2u))

Typedefs

- typedef uint32_t hsm_verification_status_t
- typedef uint8_t hsm_op_verify_sign_flags_t

Functions

- hsm_err_t hsm_verify_sign (hsm_hdl_t session_hdl, op_verify_sign_args_t *args, hsm_verification_status_t *verification_status)
- hsm_err_t hsm_open_signature_verification_service (hsm_hdl_t session_hdl, open_svc_sign_ver_args_t *args, hsm_hdl_t *signature_ver_hdl)
- hsm_err_t hsm_close_signature_verification_service (hsm_hdl_t signature_ver_hdl)
- hsm_err_t hsm_verify_signature (hsm_hdl_t signature_ver_hdl, op_verify_sign_args_t *args, hsm_verification_status_t *status)

5.5.1 Detailed Description

5.5.2 Data Structure Documentation

5.5.2.1 struct open_svc_sign_ver_args_t Structure to represent verify sign open service arguments

Data Fields

hsm_hdl_t	sig_ver_hdl	

5.5.2.2 struct op_verify_sign_args_t Structure to represent verify signature operation arguments

Data Fields

uint8_t *	key	pointer to the public key to be used for the verification. If the HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL is set, it must point to the key reference returned by the hsm_import_public_key API.
uint8_t *	message	pointer to the input (message or message digest)
uint8_t *	signature	pointer to the input signature. The signature S=(r,s) is expected to be in the format r||s||Ry where Ry is an additional byte containing the lsb of y. Ry will be considered as valid only if the HSM_OP_VERIFY_SIG↩ N_FLAGS_COMPRESSED_POINT is set.
uint16_t	key_size	length in bytes of the input key
uint16_t	signature_size	length in bytes of the output - it must contain one additional byte where to store the Ry.
uint32_t	message_size	length in bytes of the input message.
hsm_verification_status_t	verification_status	verification status.
hsm_signature_scheme_id_t	scheme_id	identifier of the digital signature scheme to be used for the operation
uint16_t	salt_len	salt length in bytes
hsm_bit_key_sz_t	key_sz	indicates key security size in bits.
hsm_pubkey_type_t	pkey_type	indicates the public key type
hsm_op_verify_sign_flags_t	flags	bitmap specifying the operation attributes

5.5.3 Macro Definition Documentation

5.5.3.1 HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST

```
#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT_D↩
IGEST ((hsm_op_verify_sign_flags_t)(0u << 0))
```

Verify signature bit indicating input is message digest

5.5.3.2 HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE

```
#define HSM_OP_VERIFY_SIGN_FLAGS_INPUT↩
_MESSAGE ((hsm_op_verify_sign_flags_t)(1u << 0))
```

Verify signature bit indicating input is actual message

5.5.3.3 HSM_OP_VERIFY_SIGN_FLAGS_COMPRESSED_POINT

```
#define HSM_OP_VERIFY_SIGN_FLAGS_↩
COMPRESSED_POINT ((hsm_op_verify_sign_flags_t)(1u << 1))
```

Verify signature bit indicating input based on signature format

5.5.3.4 HSM_OP_VERIFY_SIGN_FLAGS_KEY_INTERNAL

```
#define HSM_OP_VERIFY_SIGN_FLAGS_KEY_IN↩
TERNAL ((hsm_op_verify_sign_flags_t)(1u << 2))
```

Verify signature bit indicating input is key argument

5.5.3.5 HSM_VERIFICATION_STATUS_SUCCESS

```
#define HSM_VERIFICATION_STATUS_SUCCESS ((hsm_verification_statu
A3CC3A5u))
```

Verify signature response success status

5.5.3.6 HSM_VERIFICATION_STATUS_FAILURE

```
#define HSM_VERIFICATION_STATUS_FAILURE ((hsm_verification_status
B4DD4B2u))
```

Verify signature response failure status

5.5.4 Typedef Documentation

5.5.4.1 hsm_verification_status_t

```
typedef uint32_t hsm_verification_status_t
```

Bit indicating the response verification status

5.5.4.2 hsm_op_verify_sign_flags_t

```
typedef uint8_t hsm_op_verify_sign_flags_t
```

Bit indicating the requested operations

5.5.5 Function Documentation

5.5.5.1 hsm_verify_sign()

```
hsm_err_t hsm_verify_sign (
        hsm_hdl_t session_hdl,
        op_verify_sign_args_t * args,
        hsm_verification_status_t * verification_status )
```

Secondary API to verify a message signature.

This API does the following:

1. Open a flow for verification of the signature.

2. Based on the flag to identify the type of message: Digest or actual message, verification of the signature is done using the public key.

3. Post performing the operation, terminate the previously opened signature-verification service flow.
   User can call this function only after having opened a session.

Parameters

session_hdl	handle identifying the current key-store.
args	pointer to the structure containing the function arguments.
verification_status	pointer for storing the verification status.

Returns

error code

5.5.5.2 hsm_open_signature_verification_service()

```
hsm_err_t hsm_open_signature_verification_service (
            hsm_hdl_t session_hdl,
            open_svc_sign_ver_args_t * args,
            hsm_hdl_t * signature_ver_hdl )
```

User must open this service in order to perform signature verification operations. User can call this function only after having opened a session.

Parameters

session_hdl	handle identifying the current session.
args	pointer to the structure containing the function arguments.
signature_ver_hdl	pointer to where the signature verification service flow handle must be written.

Returns

error code

5.5.5.3 hsm_close_signature_verification_service()

```
hsm_err_t hsm_close_signature_verification_service (
            hsm_hdl_t signature_ver_hdl )
```

Terminate a previously opened signature verification service flow

Parameters

signature_ver_hdl	handle identifying the signature verification service flow to be closed.

Returns

error code

5.5.5.4 hsm_verify_signature()

```
hsm_err_t hsm_verify_signature (
            hsm_hdl_t signature_ver_hdl,
            op_verify_sign_args_t * args,
            hsm_verification_status_t * status )
```

Verify a digital signature according to the signature scheme User can call this function only after having opened a signature verification service flow.

The signature S=(r,s) is expected to be in format r||s||Ry where:

- Ry is an additional byte containing the lsb of y. Ry will be considered as valid only, if the HSM_OP_VERIF↩ Y_SIGN_FLAGS_COMPRESSED_POINT is set.

Only not-compressed keys (x,y) can be used by this command. Compressed keys can be decompressed by using the dedicated API.

In case of HSM_SIGNATURE_SCHEME_DSA_SM2_FP_256_SM3, message of op_verify_sign_args_t should be (as specified in GB/T 32918):

- equal to Z||M in case of HSM_OP_VERIFY_SIGN_FLAGS_INPUT_MESSAGE
- equal to SM3(Z||M) in case of HSM_OP_VERIFY_SIGN_FLAGS_INPUT_DIGEST

Parameters

signature_ver_hdl	handle identifying the signature verification service flow.
args	pointer to the structure containing the function arguments.
status	pointer to where the verification status must be stored if the verification succeed the value HSM_VERIFICATION_STATUS_SUCCESS is returned.

Returns

error code

5.6 Random number generation

Data Structures

- struct op_get_random_args_t

Functions

- hsm_err_t hsm_do_rng (hsm_hdl_t session_hdl, op_get_random_args_t *args)
- hsm_err_t hsm_get_random (hsm_hdl_t rng_hdl, op_get_random_args_t *args)

5.6.1 Detailed Description

5.6.2 Data Structure Documentation

5.6.2.1 struct op_get_random_args_t Structure detailing the get random number operation member arguments

Data Fields

uint8_t *	output	pointer to the output area where the random number must be written
uint32_t	random_size	length in bytes of the random number to be provided.

5.6.3 Function Documentation

5.6.3.1 hsm_do_rng()

```
hsm_err_t hsm_do_rng (
        hsm_hdl_t session_hdl,
        op_get_random_args_t * args )
```

Secondary API to fetch the Random Number
This API does the following: Get a freshly generated random number

Parameters

session_hdl	handle identifying the current session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.6.3.2 hsm_get_random() hsm_err_t hsm_get_random (hsm_hdl_t *rng_hdl*, op_get_random_args_t * *args*)

Get a freshly generated random number
User can call this function only after having opened a rng service flow

Parameters

rng_hdl	handle identifying the rng service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.7 Hashing

Data Structures

- struct op_hash_one_go_args_t

Macros

- #define HSM_HASH_FLAG_ALLOWED

Enumerations

- enum hsm_hash_algo_t {
  HSM_HASH_ALGO_SHA_224 = 0x02000008,
  HSM_HASH_ALGO_SHA_256 = 0x02000009,
  HSM_HASH_ALGO_SHA_384 = 0x0200000A,
  HSM_HASH_ALGO_SHA_512 = 0x0200000B }
- enum hsm_hash_svc_flags_t {
  HSM_HASH_FLAG_ONE_SHOT = 0x1,
  HSM_HASH_FLAG_INIT = 0x2,
  HSM_HASH_FLAG_UPDATE = 0x4,
  HSM_HASH_FLAG_FINAL = 0x8,
  HSM_HASH_FLAG_GET_CONTEXT = 0x80 }

Functions

- hsm_err_t hsm_do_hash (hsm_hdl_t session_hdl, op_hash_one_go_args_t *args)
- hsm_err_t hsm_hash_one_go (hsm_hdl_t hash_hdl, op_hash_one_go_args_t *args)

5.7.1 Detailed Description

5.7.2 Data Structure Documentation

5.7.2.1 struct op_hash_one_go_args_t Structure describing the hash one go operation arguments

Data Fields

uint8_t *	msb	pointer to the MSB of address in the requester space where buffers can be found, must be 0 until supported.
uint8_t *	ctx	pointer to the context.
uint8_t *	input	pointer to the input data to be hashed
uint8_t *	output	pointer to the output area where the resulting digest must be written
uint32_t	input_size	length in bytes of the input
uint32_t	output_size	length in bytes of the output
hsm_hash_algo_t	algo	hash algorithm to be used for the operation
hsm_hash_svc_flags_t	svc_flags	flags identifying the operation init() update(), final() or one shot operation.
uint16_t	ctx_size	size of context buffer in bytes, ignored in case of one shot operation.
uint32_t	exp_output_size	expected output digest buffer size, returned by FW in case the provided output size is incorrect.
uint16_t	context_size	expected context size to allocate in bytes, if flag Get context size is set or provided context size is incorrect.

5.7.3 Macro Definition Documentation

5.7.3.1 HSM_HASH_FLAG_ALLOWED #define HSM_HASH_FLAG_ALLOWED

Value:

```
(HSM_HASH_FLAG_ONE_SHOT | HSM_HASH_FLAG_INIT \
| HSM_HASH_FLAG_UPDATE | HSM_HASH_FLAG_FINAL \
| HSM_HASH_FLAG_GET_CONTEXT)
```

Bitmap indicating the allowed hash service operations

5.7.4 Enumeration Type Documentation

5.7.4.1 hsm_hash_algo_t enum hsm_hash_algo_t

Bitmap indicating the supported hash algorithm

5.7.4.2 hsm_hash_svc_flags_t enum hsm_hash_svc_flags_t

Bit field indicating the hash service operations

5.7.5 Function Documentation

5.7.5.1 hsm_do_hash() hsm_err_t hsm_do_hash (hsm_hdl_t *session_hdl*, op_hash_one_go_args_t * *args*)

Secondary API to digest a message.
This API does the following: Perform hash

Parameters

session_hdl	handle identifying the current session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.7.5.2 hsm_hash_one_go()

```
hsm_err_t hsm_hash_one_go (
        hsm_hdl_t hash_hdl,
        op_hash_one_go_args_t * args )
```

Perform the hash operation on a given input
User can call this function only after having opened a hash service flow

Parameters

hash_hdl	handle identifying the hash service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.8 Data storage

Data Structures

- struct open_svc_data_storage_args_t
- struct op_data_storage_args_t
- struct op_enc_data_storage_args_t

Macros

- #define **HSM_OP_DATA_STORAGE_FLAGS_EL2GO** ((hsm_op_data_storage_flags_t)(1u << 0))
- #define HSM_OP_DATA_STORAGE_FLAGS_DEFAULT ((hsm_op_data_storage_flags_t)(0u << 0))

  Store data.
- #define HSM_OP_DATA_STORAGE_FLAGS_STORE ((hsm_op_data_storage_flags_t)(1u << 1))

  Retrieve data.
- #define **HSM_OP_DATA_STORAGE_FLAGS_RETRIEVE** ((hsm_op_data_storage_flags_t)(0u << 1))
- #define ENC_DATA_TLV_DEV_UUID_TAG 0x41u
- #define **ENC_DATA_TLV_IV_TAG** 0x45u
- #define **ENC_DATA_TLV_ENC_DATA_TAG** 0x46u
- #define **ENC_DATA_TLV_SIGN_TAG** 0x5Eu
- #define ENC_DATA_TLV_DEV_UUID_TAG_LEN 0x01u
- #define **ENC_DATA_TLV_IV_TAG_LEN** 0x01u
- #define **ENC_DATA_TLV_ENC_DATA_TAG_LEN** 0x01u
- #define **ENC_DATA_TLV_SIGN_TAG_LEN** 0x01u
- #define HSM_OP_ENC_DATA_STORAGE_FLAGS_RANDOM_IV ((hsm_op_enc_data_storage_flags_t)(1u << 0))

  internally generate random IV, if needed for operation.
- #define HSM_OP_ENC_DATA_STORAGE_FLAGS_READ_ONCE ((hsm_op_enc_data_storage_flags_t)(1u << 1))

  read once, and delete data from NVM after retrieve.

Typedefs

- typedef uint8_t hsm_svc_data_storage_flags_t
- typedef uint8_t hsm_op_data_storage_flags_t
- typedef uint16_t hsm_op_enc_data_storage_flags_t

Functions

- hsm_err_t hsm_data_ops (hsm_hdl_t key_store_hdl, op_data_storage_args_t *args)
- hsm_err_t hsm_enc_data_ops (hsm_hdl_t key_store_hdl, op_enc_data_storage_args_t *args)
- hsm_err_t hsm_open_data_storage_service (hsm_hdl_t key_store_hdl, open_svc_data_storage_args_t *args, hsm_hdl_t *data_storage_hdl)
- hsm_err_t hsm_data_storage (hsm_hdl_t data_storage_hdl, op_data_storage_args_t *args)
- hsm_err_t hsm_enc_data_storage (hsm_hdl_t data_storage_hdl, op_enc_data_storage_args_t *args)
- uint8_t decode_enc_data_tlv (op_data_storage_args_t *args)
- hsm_err_t hsm_close_data_storage_service (hsm_hdl_t data_storage_hdl)

5.8.1 Detailed Description

5.8.2 Data Structure Documentation

5.8.2.1 struct open_svc_data_storage_args_t Structure specifying the data storage open service member arguments

Data Fields

hsm_hdl_t	data_storage_handle	data storage handle.
hsm_svc_data_storage_flags_t	flags	bitmap specifying the services properties.
uint8_t	reserved[3]	

5.8.2.2 struct op_data_storage_args_t Structure detailing the data storage operation member arguments

Data Fields

uint8_t *	data	pointer to the data. In case of store request, it will be the input data to store. In case of retrieve, it will be the pointer where to load data.
uint32_t	data_size	length in bytes of the data
uint32_t	data_id	id of the data
hsm_op_data_storage_flags_t	flags	flags bitmap specifying the operation attributes.
hsm_svc_data_storage_flags_t	svc_flags	bitmap specifying the services properties.
uint16_t	uuid_len	Device UUID length in bytes. In case RETRIEVE, if the data retrieved is in TLV format which was stored by Encrypted Data Storage API. The TLV format data will be decoded to fill the following fields. Memory for storing uuid/iv/ciphertext/payload/signature will be allocated by HSM library. Caller of the function decode_enc_data_tlv(), needs to ensure freeing up memory.
uint8_t *	uuid	Device UUID.
uint16_t	iv_len	IV length in bytes, if needed, otherwise 0.
uint8_t *	iv	IV buffer, if needed.
uint32_t	ciphertext_len	encrypted text length in bytes
uint8_t *	ciphertext	encrypted text buffer
uint32_t	payload_len	payload length in bytes
uint8_t *	payload	payload data buffer to verify signature
uint16_t	signature_len	signature length in bytes
uint8_t *	signature	signature buffer
uint32_t	exp_output_size	expected output buffer size in bytes, valid in case of HSM_OUT_TOO_SMALL (0x1D) error code

Data Fields

uint32_t	data_id	id of the data
uint8_t *	data	pointer to the data, to be encrypted and signed
uint32_t	data_size	length in bytes of the data
uint32_t	enc_algo	cipher algorithm to be used for encryption of data
uint32_t	enc_key_id	identifier of the key to be used for encryption
uint32_t	sign_algo	signature algorithm to be used for signing the data
uint32_t	sign_key_id	identifier of the key to be used for signing
uint8_t *	iv	pointer to the IV buffer
uint16_t	iv_size	IV size in bytes.

Data Fields

hsm_op_enc_data_storage_flags_t	flags	bitmap specifying the operation attributes
hsm_svc_data_storage_flags_t	svc_flags	bitmap specifying the service attributes.
uint16_t	lifecycle	bitmask of device lifecycle, in which the data can be retrieved
uint32_t	out_data_size	size (bytes) of the signed TLV stored, received with API resp

5.8.2.3 struct op_enc_data_storage_args_t

5.8.3 Macro Definition Documentation

5.8.3.1 ENC_DATA_TLV_DEV_UUID_TAG `#define ENC_DATA_TLV_DEV_UUID_TAG 0x41u`

Encrypted Data TLV Tags

5.8.3.2 ENC_DATA_TLV_DEV_UUID_TAG_LEN `#define ENC_DATA_TLV_DEV_UUID_TAG_LEN 0x01u`

Encrypted Data TLV Tags lengths

5.8.4 Typedef Documentation

5.8.4.1 hsm_svc_data_storage_flags_t `typedef uint8_t hsm_svc_data_storage_flags_t`

Bitmap specifying the data storage open service supported properties

5.8.4.2 hsm_op_data_storage_flags_t `typedef uint8_t hsm_op_data_storage_flags_t`

Bitmap specifying the data storage operation supported attributes

5.8.4.3 hsm_op_enc_data_storage_flags_t `typedef uint16_t hsm_op_enc_data_storage_flags_t`

Bitmap specifying the encrypted data storage operation supported attributes

5.8.5 Function Documentation

5.8.5.1 hsm_data_ops() hsm_err_t hsm_data_ops (
hsm_hdl_t *key_store_hdl*,
op_data_storage_args_t * *args*)

Secondary API to store and retrieve data from the linux filesystem managed by EdgeLock Enclave Firmware.

This API does the following:

1. Open an data storage service Flow
2. Based on the flag for operation attribute: Store or Retrieve,
   - Store the data
   - Retrieve the data, from the non-volatile storage.
3. Post performing the operation, terminate the previously opened data-storage service flow.
   User can call this function only after having opened a key-store.

Parameters

key_store_hdl	handle identifying the current key-store.
args	pointer to the structure containing the function arguments.

Returns

error code

5.8.5.2 hsm_enc_data_ops() hsm_err_t hsm_enc_data_ops (
hsm_hdl_t *key_store_hdl*,
op_enc_data_storage_args_t * *args*)

Secondary API to store encrypted and signed data in NVM.

This API does the following:

1. Open an data storage service Flow
2. Store the encryted and signed data in NVM. The stored data can be retrieved through Data Storage API
3. Post performing the operation, terminate the previously opened
   data-storage service flow.
   User can call this function only after having opened a key-store.

Parameters

key_store_hdl	handle identifying the current key-store.
args	pointer to the structure containing the function arguments.

Returns

error code

5.8.5.3 hsm_open_data_storage_service()

```
hsm_err_t hsm_open_data_storage_service (
    hsm_hdl_t key_store_hdl,
    open_svc_data_storage_args_t * args,
    hsm_hdl_t * data_storage_hdl )
```

Open a data storage service flow
User must open this service flow in order to store/retrieve generic data in/from the HSM.

Parameters

key_store_hdl	handle identifying the key store service flow.
args	pointer to the structure containing the function arguments.
data_storage_hdl	pointer to where the data storage service flow handle must be written.

Returns

error_code error code.

5.8.5.4 hsm_data_storage()

```
hsm_err_t hsm_data_storage (
    hsm_hdl_t data_storage_hdl,
    op_data_storage_args_t * args )
```

Store or retrieve generic data identified by a data_id.

Parameters

data_storage_hdl	handle identifying the data storage service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.8.5.5 hsm_enc_data_storage()

```
hsm_err_t hsm_enc_data_storage (
    hsm_hdl_t data_storage_hdl,
    op_enc_data_storage_args_t * args )
```

Store encrypted and signed data in the NVM.

Parameters

data_storage_hdl	handle identifying the data storage service flow.
args	pointer to the structure containing the function arguments.

Returns

error code

5.8.5.6 decode_enc_data_tlv() `uint8_t decode_enc_data_tlv (op_data_storage_args_t * args )`

Decode and populate the data storage op args for Encrypted Data TLV fields

Parameters

args	pointer to the structure containing Retrieved Encrypted Data TLV buffer and to be populated with decoded data from TLV.

Returns

error code 0 for success

5.8.5.7 hsm_close_data_storage_service() `hsm_err_t hsm_close_data_storage_service (hsm_hdl_t data_storage_hdl )`

Terminate a previously opened data storage service flow

Parameters

data_storage_hdl	handle identifying the data storage service flow.

Returns

error code

5.9 Authenticated Encryption

Functions

- hsm_err_t hsm_do_auth_enc (hsm_hdl_t key_store_hdl, op_auth_enc_args_t *auth_enc_args)

5.9.1 Detailed Description

5.9.2 Function Documentation

5.9.2.1 hsm_do_auth_enc()

```
hsm_err_t hsm_do_auth_enc (
        hsm_hdl_t key_store_hdl,
        op_auth_enc_args_t * auth_enc_args )
```

Secondary API to perform Authenticated Encryption
This API does the following:

1. Opens Cipher Service Flow
2. Perform Authenticated Encryption operation
3. Terminates the previously opened Cipher service flow
   User can call this function only after having opened a key store service flow.

Parameters

key_store_hdl	handle identifying the key store service flow.
auth_enc_args	pointer to the structure containing the function arguments.

Returns

error code

5.10 Mac

Data Structures

- struct open_svc_mac_args_t
- struct op_mac_one_go_args_t

Macros

- #define HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION ((hsm_op_mac_one_go_flags_t)(0u << 0))
- #define HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION ((hsm_op_mac_one_go_flags_t)(1u << 0))
- #define HSM_MAC_VERIFICATION_STATUS_SUCCESS ((hsm_mac_verification_status_t)(0x6C1AA1↩ C6u))

Typedefs

- typedef uint8_t hsm_op_mac_one_go_flags_t
- typedef uint32_t hsm_mac_verification_status_t
- typedef hsm_permitted_algo_t hsm_op_mac_one_go_algo_t

Functions

- hsm_err_t hsm_do_mac (hsm_hdl_t key_store_hdl, op_mac_one_go_args_t *mac_one_go)
- hsm_err_t hsm_open_mac_service (hsm_hdl_t key_store_hdl, open_svc_mac_args_t *args, hsm_hdl_t *mac_hdl)
- hsm_err_t hsm_mac_one_go (hsm_hdl_t mac_hdl, op_mac_one_go_args_t *args, hsm_mac_verification_status_t *status)
- hsm_err_t hsm_close_mac_service (hsm_hdl_t mac_hdl)

5.10.1 Detailed Description

5.10.2 Data Structure Documentation

5.10.2.1 struct open_svc_mac_args_t Structure describing the mac open service member agruments

Data Fields

hsm_hdl_t	mac_serv_hdl	indicates the mac handle.

5.10.2.2 struct op_mac_one_go_args_t Structure describing the mac one go operation member agruments

Data Fields

uint32_t	key_identifier	identifier of the key to be used for the operation
hsm_op_mac_one_go_algo_t	algorithm	algorithm to be used for the operation

Data Fields

hsm_op_mac_one_go_flags_t	flags	bitmap specifying the operation attributes
uint8_t *	payload	pointer to the payload area
uint8_t *	mac	pointer to the tag area
uint32_t	payload_size	length in bytes of the payload
uint16_t	mac_size	length of the tag. • Specified in bytes if HSM_OP_MAC_ONE_GO_FLAGS_MAC_LENGTH_IN_BITS is clear. • Specified in bits when HSM_OP_MAC_ONE_GO_FLAGS_MAC_LENGTH_IN_BITS is set. Note: – When specified in bytes the mac size cannot be less than 4 bytes. – When specified in bits the mac size cannot be less than: – the key specific min_mac_len setting if specified for this key when generated/injected; or – the min_mac_length value if specified at the key store provisioning. (if a key specific setting was not specified at key generation/injection); or – the default value (32 bit) if a minimum has not been specified using one of the above 2 methods.
hsm_mac_verification_status_t	verification_status	mac verification status.
uint16_t	exp_mac_size	expected mac size for output, returned by FW in case the mac size provided is less than the expected mac size calculated from MAC algorithm.

5.10.3 Macro Definition Documentation

5.10.3.1 HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION

```
#define HSM_OP_MAC_ONE_GO_FLAGS_MAC_VERIFICATION ((hsm_op_mac_one_go_flags_t)(0u << 0))
```

Bit indicating mac one go verify operation

5.10.3.2 HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION

```
#define HSM_OP_MAC_ONE_GO_FLAGS_MAC_GENERATION ((hsm_op_mac_one_go_flags_t)(1u << 0))
```

Bit indicating mac one go generate operation

5.10.3.3 HSM_MAC_VERIFICATION_STATUS_SUCCESS

```
#define HSM_MAC_VERIFICATION_STATUS_SUCCESS ((hsm_mac_verification_status_t)(0x6C1AA1C6u))
```

Bit indicating mac verification success status

5.10.4 Typedef Documentation

5.10.4.1 hsm_op_mac_one_go_flags_t `typedef uint8_t hsm_op_mac_one_go_flags_t`

Bitmap describing the mac one go operation

5.10.4.2 hsm_mac_verification_status_t `typedef uint32_t hsm_mac_verification_status_t`

Bitmap describing the mac verification status

5.10.4.3 hsm_op_mac_one_go_algo_t `typedef hsm_permitted_algo_t hsm_op_mac_one_go_algo_t`

Bitmap describing the mac one go operation permitted algorithm < Following three permitted algos are allowed:

- PERMITTED_ALGO_HMAC_SHA256 = 0x03800009,
- PERMITTED_ALGO_HMAC_SHA384 = 0x0380000A,
  - PERMITTED_ALGO_CMAC = 0x03C00200,

5.10.5 Function Documentation

5.10.5.1 hsm_do_mac()

```
hsm_err_t hsm_do_mac (
        hsm_hdl_t key_store_hdl,
        op_mac_one_go_args_t * mac_one_go )
```

Secondary API to perform mac operation
This API does the following:

1. Open an MAC Service Flow
2. Perform mac operation
3. Terminate a previously opened mac service flow
   User can call this function only after having opened a key store service flow.

Parameters

key_store_hdl	handle identifying the key store service flow.
mac_one_go	pointer to the structure containing the function arguments.

Returns

error code

5.10.5.2 hsm_open_mac_service()

```
hsm_err_t hsm_open_mac_service (
    hsm_hdl_t key_store_hdl,
    open_svc_mac_args_t * args,
    hsm_hdl_t * mac_hdl )
```

Open a mac service flow

User can call this function only after having opened a key store service flow.
User must open this service in order to perform mac operation

Parameters

key_store_hdl	handle identifying the key store service flow.
args	pointer to the structure containing the function arguments.
mac_hdl	pointer to where the mac service flow handle must be written.

Returns

error code

5.10.5.3 hsm_mac_one_go()

```
hsm_err_t hsm_mac_one_go (
    hsm_hdl_t mac_hdl,
    op_mac_one_go_args_t * args,
    hsm_mac_verification_status_t * status )
```

Perform mac operation
User can call this function only after having opened a mac service flow
For CMAC algorithm, a key of type HSM_KEY_TYPE_AES_XXX must be used
For HMAC algorithm, a key of type HSM_KEY_TYPE_HMAC_XXX must be used
For mac verification operations, the verified mac length can be specified in:

- Bits by setting the HSM_OP_MAC_ONE_GO_FLAGS_MAC_LENGTH_IN_BITS flag,
- if this flag is clear then the mac_length is specified in bytes.

For mac generation operations:

- mac length must be set in bytes, and
- HSM_OP_MAC_ONE_GO_FLAGS_MAC_LENGTH_IN_BITS flag must be 0

Parameters

mac_hdl	handle identifying the mac service flow.
args	pointer to the structure containing the function arguments.
status	pointer for storing the verification status.

Returns

error code

5.10.5.4 hsm_close_mac_service() `hsm_err_t hsm_close_mac_service ( hsm_hdl_t mac_hdl )`

Terminate a previously opened mac service flow

Parameters

mac_hdl	pointer to handle identifying the mac service flow to be closed.

Returns

error code

5.11 Dump Firmware Log

Data Structures

- struct op_debug_dump_args_t

Functions

- hsm_err_t dump_firmware_log (hsm_hdl_t session_hdl)

5.11.1 Detailed Description

5.11.2 Data Structure Documentation

5.11.2.1 struct op_debug_dump_args_t Structure detailing the debug dump operation member arguments

Data Fields

bool	is_dump_pending	
uint32_t	dump_buf_len	
uint32_t	dump_buf[MAC_BUFF_LEN]	

5.11.3 Function Documentation

5.11.3.1 dump_firmware_log() hsm_err_t dump_firmware_log (hsm_hdl_t *session_hdl*)

This command is designed to dump the firmware logs

Parameters

session_hdl	handle identifying the session handle.

Returns

error code

5.12 Dev attest

Data Structures

- struct op_dev_attest_args_t

Macros

- #define DEV_ATTEST_NOUNCE_SIZE_V1 (4)
- #define **DEV_ATTEST_NOUNCE_SIZE_V2** (16)

Functions

- hsm_err_t hsm_dev_attest (hsm_hdl_t sess_hdl, op_dev_attest_args_t *args)

5.12.1 Detailed Description

5.12.2 Data Structure Documentation

5.12.2.1 struct op_dev_attest_args_t Structure describing the device attestation operation member arguments Memory for storing uid/sha_rom_patch/sha_fw/signature will be allocated by HSM library. Caller of the func hsm_dev_attest(), needs to ensure freeing up memory.

Data Fields

uint16_t	soc_id	SoC ID.
uint16_t	soc_rev	SoC Revision.
uint16_t	lmda_val	Lmda Lifecycle value.
uint8_t	ssm_state	Security Subsystem State Machine state.
uint8_t	uid_sz	buffer size in bytes for Chip Unique Identifier
uint8_t *	uid	pointer to the Chip Unique Identifier buffer
uint16_t	rom_patch_sha_sz	buffer size in bytes for SHA256 of Sentinel ROM patch fuses
uint16_t	sha_fw_sz	buffer size in bytes for first 256 bits of installed FW SHA
uint8_t *	sha_rom_patch	pointer to the buffer containing SHA256 of Sentinel ROM patch fuses
uint8_t *	sha_fw	pointer to the buffer containing first 256 bits of installed FW SHA
uint16_t	nounce_sz	buffer size in bytes for request nounce value
uint8_t *	nounce	pointer to the input/request nounce value buffer
uint16_t	rsp_nounce_sz	size in bytes for FW nounce buffer, returned with FW resp
uint8_t *	rsp_nounce	pointer to the FW nounce buffer, returned with FW resp
uint16_t	oem_srkh_sz	buffer size in bytes for OEM SRKH (version 2)
uint8_t *	oem_srkh	pointer to the buffer of OEM SRKH (version 2)
uint8_t	imem_state	IMEM state (version 2)
uint8_t	csal_state	CSAL state (version 2)
uint8_t	trng_state	TRNG state (version 2)
uint16_t	info_buf_sz	size in bytes for info buffer
uint8_t *	info_buf	pointer to the info buffer, for verification of the signature
uint8_t	attest_result	Attest Result. 0 means pass. 1 means fail.
uint16_t	sign_sz	buffer size in bytes for signature
uint8_t *	signature	pointer to the signature buffer

5.12.3 Macro Definition Documentation

5.12.3.1 DEV_ATTEST_NOUNCE_SIZE_V1 `#define DEV_ATTEST_NOUNCE_SIZE_V1 (4)`

Device Attestation Nounce sizes

5.12.4 Function Documentation

5.12.4.1 hsm_dev_attest() `hsm_err_t hsm_dev_attest ( hsm_hdl_t sess_hdl, op_dev_attest_args_t * args )`

Perform device attestation operation
User can call this function only after having opened the session.

Parameters

sess_hdl	handle identifying the active session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.13 Dev Info

Data Structures

- struct op_dev_getinfo_args_t

Functions

- hsm_err_t hsm_dev_getinfo (hsm_hdl_t sess_hdl, op_dev_getinfo_args_t *args)

5.13.1 Detailed Description

5.13.2 Data Structure Documentation

5.13.2.1 struct op_dev_getinfo_args_t Structure detailing the device getinfo operation member arguments Memory for storing uid/sha_rom_patch/sha_fw/signature will be allocated by HSM library. Caller of the func hsm_dev_getinfo(), needs to ensure freeing up memory.

Data Fields

uint16_t	soc_id	SoC ID.
uint16_t	soc_rev	SoC revision number.
uint16_t	lmda_val	indicates the lmda lifecycle value.
uint8_t	ssm_state	security subsystem state machine.
uint8_t	uid_sz	chip unique identifier size.
uint8_t *	uid	pointer to the chip unique identifier.
uint16_t	rom_patch_sha_sz	indicates the size of Sha256 of sentinel rom patch fuses.
uint16_t	sha_fw_sz	indicates the size of first 256 bits of installed fw sha.
uint8_t *	sha_rom_patch	pointer to the Sha256 of sentinel rom patch fuses digest.
uint8_t *	sha_fw	pointer to the first 256 bits of installed fw sha digest.
uint16_t	oem_srkh_sz	indicates the size of FW OEM SRKH.
uint8_t *	oem_srkh	pointer to the FW OEM SRKH.
uint8_t	imem_state	indicates the imem state.
uint8_t	csal_state	crypto Lib random context initialization state.
uint8_t	trng_state	indicates TRNG state.

5.13.3 Function Documentation

5.13.3.1 hsm_dev_getinfo()

```
hsm_err_t hsm_dev_getinfo (
        hsm_hdl_t sess_hdl,
        op_dev_getinfo_args_t * args )
```

Perform device attestation operation
User can call this function only after having opened the session.

Parameters

sess_hdl	handle identifying the active session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.14 Generic Crypto: Asymmetric Crypto

Data Structures

- struct op_gc_acrypto_args_t

Macros

- #define HSM_OP_GC_ACRYPTO_FLAGS_INPUT_MESSAGE ((hsm_op_gc_acrypto_flags_t)(1u << 0))
- #define HSM_GC_ACRYPTO_VERIFICATION_SUCCESS ((hsm_gc_acrypto_verification_status_t)(0x5↩ A3CC3A5u))
- #define HSM_GC_ACRYPTO_VERIFICATION_FAILURE ((hsm_gc_acrypto_verification_status_t)(0x2B4↩ DD4B2u))

Typedefs

- typedef uint8_t hsm_op_gc_acrypto_flags_t
- typedef uint32_t hsm_gc_acrypto_verification_status_t

Enumerations

- enum hsm_op_gc_acrypto_algo_t {
  HSM_GC_ACRYPTO_ALGO_ECDSA_SHA224 = ALGO_ECDSA_SHA224,
  HSM_GC_ACRYPTO_ALGO_ECDSA_SHA256 = ALGO_ECDSA_SHA256,
  HSM_GC_ACRYPTO_ALGO_ECDSA_SHA384 = ALGO_ECDSA_SHA384,
  HSM_GC_ACRYPTO_ALGO_ECDSA_SHA512 = ALGO_ECDSA_SHA512,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA224 = ALGO_RSA_PKCS1_V15_SHA224,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA256 = ALGO_RSA_PKCS1_V15_SHA256,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA384 = ALGO_RSA_PKCS1_V15_SHA384,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_SHA512 = ALGO_RSA_PKCS1_V15_SHA512,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA224 = ALGO_RSA_PKCS1_PSS_MGF1_↩ SHA224,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA256 = ALGO_RSA_PKCS1_PSS_MGF1_↩ SHA256,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA384 = ALGO_RSA_PKCS1_PSS_MGF1_↩ SHA384,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_PSS_MGF1_SHA512 = ALGO_RSA_PKCS1_PSS_MGF1_↩ SHA512,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_V15_CRYPT = ALGO_RSA_PKCS1_V15_CRYPT,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA1 = ALGO_RSA_PKCS1_OAEP_SHA1,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA224 = ALGO_RSA_PKCS1_OAEP_SHA224,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA256 = ALGO_RSA_PKCS1_OAEP_SHA256,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA384 = ALGO_RSA_PKCS1_OAEP_SHA384,
  HSM_GC_ACRYPTO_ALGO_RSA_PKCS1_OAEP_SHA512 = ALGO_RSA_PKCS1_OAEP_SHA512 }
- enum hsm_gc_acrypto_op_mode_t {
  HSM_GC_ACRYPTO_OP_MODE_ENCRYPT = 0x01,
  HSM_GC_ACRYPTO_OP_MODE_DECRYPT = 0x02,
  HSM_GC_ACRYPTO_OP_MODE_SIGN_GEN = 0x03,
  HSM_GC_ACRYPTO_OP_MODE_SIGN_VER = 0x04 }

Functions

- hsm_err_t hsm_gc_acrypto (hsm_hdl_t session_hdl, op_gc_acrypto_args_t *args)

5.14.1 Detailed Description

5.14.2 Data Structure Documentation

5.14.2.1 struct op_gc_acrypto_args_t

Structure describing the generic asymmetric crypto member arguments

Data Fields

hsm_op_gc_acrypto_algo_t	algorithm	algorithm to use for the operation
hsm_gc_acrypto_op_mode_t	op_mode	indicates the operation mode
hsm_op_gc_acrypto_flags_t	flags	indicates operation flags
hsm_bit_key_sz_t	bit_key_sz	key size in bits
uint8_t *	data_buff1	pointer to the data buffer 1: • plaintext in case of encryption/decryption op • digest or message in case of signature generation/verification op
uint8_t *	data_buff2	pointer to the data buffer 2: • ciphertext in case of encryption/decryption op • signature in case of signature generation/verification op
uint32_t	data_buff1_size	size in bytes of data buffer 1
uint32_t	data_buff2_size	size in bytes of data buffer 2
uint8_t *	key_buff1	pointer to the key modulus buffer
uint8_t *	key_buff2	pointer the key exponent, either private or public -Encryption mode, public exponent -Decryption mode, private exponent -Signature Generation mode, private exponent -Signature Verification mode, public exponent
uint16_t	key_buff1_size	size in bytes of the key buffer 1
uint16_t	key_buff2_size	size in bytes of the key buffer 2
uint8_t *	rsa_label	RSA label address -only used for OAEP encryption/decryption op mode and optional
uint16_t	rsa_label_size	RSA label size in bytes -only used for OAEP encryption/decryption op mode
uint16_t	rsa_salt_len	RSA salt length in bytes -only used for PSS signature algorithm scheme
uint32_t	exp_plaintext_len	expected plaintext length in bytes, returned by FW in case of DECRYPT operation mode
hsm_gc_acrypto_verification_status_t	verification_status	signature verification status

5.14.3 Macro Definition Documentation

5.14.3.1 HSM_OP_GC_ACRYPTO_FLAGS_INPUT_MESSAGE

```
#define HSM_OP_GC_ACRYPTO_FLAGS_INPU↩
T_MESSAGE ((hsm_op_gc_acrypto_flags_t)(1u << 0))
```

Bit indicating the generic asymmetric crypto input message operation

5.14.3.2 HSM_GC_ACRYPTO_VERIFICATION_SUCCESS

```
#define HSM_GC_ACRYPTO_VERIFICATION_SUCC↩
ESS ((hsm_gc_acrypto_verification_status_t)(0x5A3CC3A5u))
```

Bit indicating the generic asymmetric crypto success verification status

5.14.3.3 HSM_GC_ACRYPTO_VERIFICATION_FAILURE

```
#define HSM_GC_ACRYPTO_VERIFICATION_FAILU↩
RE ((hsm_gc_acrypto_verification_status_t)(0x2B4DD4B2u))
```

Bit indicating the generic asymmetric crypto failure verification status

5.14.4 Typedef Documentation

5.14.4.1 hsm_op_gc_acrypto_flags_t

```
typedef uint8_t hsm_op_gc_acrypto_flags_t
```

Bitmap describing the generic asymmetric crypto supported operation

5.14.4.2 hsm_gc_acrypto_verification_status_t

```
typedef uint32_t hsm_gc_acrypto_verification_status_t
```

Bitmap describing the generic asymmetric crypto verification status

5.14.5 Enumeration Type Documentation

5.14.5.1 hsm_op_gc_acrypto_algo_t

```
enum hsm_op_gc_acrypto_algo_t
```

Enum detailing the generic asymmetric crypto supported algorithms

5.14.5.2 hsm_gc_acrypto_op_mode_t

```
enum hsm_gc_acrypto_op_mode_t
```

Enum describing the generic asymmetric crypto supported operating modes

5.14.6 Function Documentation

5.14.6.1 hsm_gc_acrypto()

```
hsm_err_t hsm_gc_acrypto (
        hsm_hdl_t session_hdl,
        op_gc_acrypto_args_t * args )
```

This command is designed to perform the following operations: -Asymmetric crypto -encryption/decryption -signature generation/verification

Parameters

session_hdl	handle identifying the current session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.15 Generic Crypto Asymmetric Key Generate

Data Structures

- struct op_gc_akey_gen_args_t

Functions

- hsm_err_t hsm_gc_akey_gen (hsm_hdl_t session_hdl, op_gc_akey_gen_args_t *args)

5.15.1 Detailed Description

5.15.2 Data Structure Documentation

5.15.2.1 struct op_gc_akey_gen_args_t Structue detailing the generic crypto asymmetric key generate operation members

Data Fields

uint8_t *	modulus	pointer to the output buffer of key modulus
uint8_t *	priv_buff	pointer to the output buffer of key private exponent
uint8_t *	pub_buff	pointer to the input buffer containing key public exponent
uint16_t	modulus_size	size in bytes of the modulus buffer
uint16_t	priv_buff_size	size in bytes of the private exponent buffer
uint16_t	pub_buff_size	size in bytes of the public exponent buffer
hsm_key_type_t	key_type	indicates which type of keypair must be generated
hsm_bit_key_sz_t	bit_key_sz	size in bits of the keypair to be generated

5.15.3 Function Documentation

5.15.3.1 hsm_gc_akey_gen()

```
hsm_err_t hsm_gc_akey_gen (
    hsm_hdl_t session_hdl,
    op_gc_akey_gen_args_t * args )
```

This command is designed to perform the following operations: -Generate asymmetric keys, without using FW keystore

Parameters

session_hdl	handle identifying the current session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.16 Get Info

Data Structures

- struct op_get_info_args_t

Functions

- hsm_err_t hsm_get_info (hsm_hdl_t sess_hdl, op_get_info_args_t *args)

5.16.1 Detailed Description

5.16.2 Data Structure Documentation

5.16.2.1 struct op_get_info_args_t Structure describing the get info operation member arguments

Data Fields

uint32_t	user_sab_id	Stores User identifier (32bits)
uint8_t *	chip_unique_id	Stores the chip unique identifier.
uint16_t	chip_unq_id_sz	Size of the chip unique identifier in bytes.
uint16_t	chip_monotonic_counter	Stores the chip monotonic counter value (16bits)
uint16_t	chip_life_cycle	Stores the chip current life cycle bitfield (16bits)
uint32_t	version	Stores the module version (32bits)
uint32_t	version_ext	Stores the module extended version (32bits)
uint8_t	fips_mode	Stores the FIPS mode bitfield (8bits). Bitmask definition: bit0 - FIPS mode of operation: • value 0 - part is running in FIPS non-approved mode. • value 1 - part is running in FIPS approved mode. bit1 - FIPS certified part: • value 0 - part is not FIPS certified. • value 1 - part is FIPS certified. bit2-7: reserved • value 0.

5.16.3 Function Documentation

5.16.3.1 hsm_get_info()

```
hsm_err_t hsm_get_info (
        hsm_hdl_t sess_hdl,
        op_get_info_args_t * args )
```

Perform device attestation operation
User can call this function only after having opened the session.

Parameters

sess_hdl	handle identifying the active session.
args	pointer to the structure containing the function arguments.

Returns

error code

5.17 Public key recovery

Public Key Recovery is now also known as Public Key Exportation, in PSA compliant APIs. The naming here has been kept unchanged, for backward compatibility and Non-PSA compliant APIs.
.

Data Structures

- struct op_pub_key_recovery_args_t

Functions

- hsm_err_t hsm_pub_key_recovery (hsm_hdl_t key_store_hdl, op_pub_key_recovery_args_t *args)

5.17.1 Detailed Description

Public Key Recovery is now also known as Public Key Exportation, in PSA compliant APIs. The naming here has been kept unchanged, for backward compatibility and Non-PSA compliant APIs.
.

5.17.2 Data Structure Documentation

5.17.2.1 struct op_pub_key_recovery_args_t Structure detailing the public key recovery opeation member arguments

Data Fields

uint32_t	key_identifier	< pointer to the identifier of the key to be used for the operation pointer to the output area where the generated public key must be written
uint8_t *	out_key	length in bytes of the output key
uint16_t	out_key_size	expected output key buffer size, valid in case of HSM_OUT_TOO_SMALL
uint16_t	exp_out_key_size	

5.17.3 Function Documentation

5.17.3.1 hsm_pub_key_recovery()

```
hsm_err_t hsm_pub_key_recovery (
        hsm_hdl_t key_store_hdl,
        op_pub_key_recovery_args_t * args )
```

Recover Public key from private key present in key store
User can call this function only after having opened a key store.

Parameters

key_store_hdl	handle identifying the current key store.
args	pointer to the structure containing the function arguments.

Returns

error code

5.18 Key store

User must open a key store service flow in order to perform the following operations:

Data Structures

- struct open_svc_key_store_args_t

Macros

- #define HSM_SVC_KEY_STORE_FLAGS_LOAD ((hsm_svc_key_store_flags_t)(0u << 0))
  It must be specified to load a previously created key store.
- #define HSM_SVC_KEY_STORE_FLAGS_CREATE ((hsm_svc_key_store_flags_t)(1u << 0))
- #define HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN ((hsm_svc_key_store_flags_t)(1u << 3))
- #define HSM_SVC_KEY_STORE_FLAGS_STRICT_OPERATION ((hsm_svc_key_store_flags_t)(1u << 7))

Typedefs

- typedef uint8_t hsm_svc_key_store_flags_t

Functions

- hsm_err_t hsm_open_key_store_service (hsm_hdl_t session_hdl, open_svc_key_store_args_t *args, hsm_hdl_t *key_store_hdl)
- hsm_err_t hsm_close_key_store_service (hsm_hdl_t key_store_hdl)

5.18.1 Detailed Description

User must open a key store service flow in order to perform the following operations:

- create a new key store
- perform operations involving keys stored in the key store (ciphering, signature generation...)
- perform a key store reprovisioning using a signed message. A key store re-provisioning results in erasing all the key stores handled by the HSM.

To grant access to the key store, the caller is authenticated against the domain ID (DID) and Messaging Unit used at the keystore creation, additionally an authentication nonce can be provided.

5.18.2 Data Structure Documentation

5.18.2.1 struct open_svc_key_store_args_t Structure specifying the open key store service member arguments

Data Fields

uint32_t	key_store_hdl	handle identifying the key store service flow
uint32_t	key_store_identifier	user defined id identifying the key store. Only one key store service can be opened on a given key_store_identifier.
uint32_t	authentication_nonce	user defined nonce used as authentication proof for accessing the key store.
uint8_t	flags	bitmap specifying the services properties.
uint8_t *	signed_message	pointer to signed_message to be sent only in case of key store re-provisioning.
uint16_t	signed_msg_size	size of the signed_message to be sent only in case of key store re-provisioning.

5.18.3 Macro Definition Documentation

5.18.3.1 HSM_SVC_KEY_STORE_FLAGS_CREATE `#define HSM_SVC_KEY_STORE_FLAGS_CREATE ((hsm_svc_key_store_fla << 0))`

It must be specified to create a new key store. The key store will be stored in the NVM only if the STRICT OPER↩ ATION flag is set.

5.18.3.2 HSM_SVC_KEY_STORE_FLAGS_SET_MAC_LEN `#define HSM_SVC_KEY_STORE_FLAGS_SET_MAC↩ _LEN ((hsm_svc_key_store_flags_t)(1u << 3))`

If set, minimum mac length specified in min_mac_length field will be stored in the key store when creating the key store. Must only be set at key store creation.

5.18.3.3 HSM_SVC_KEY_STORE_FLAGS_STRICT_OPERATION `#define HSM_SVC_KEY_STORE_FLAGS_ST↩ RICT_OPERATION ((hsm_svc_key_store_flags_t)(1u << 7))`

The request is completed only when the new key store has been written in in the NVM. This applicable for CREATE operations only.

5.18.4 Typedef Documentation

5.18.4.1 hsm_svc_key_store_flags_t `typedef uint8_t hsm_svc_key_store_flags_t`

Bitmap specifying the open key store service supported attributes

5.18.5 Function Documentation

5.18.5.1 hsm_open_key_store_service() `hsm_err_t hsm_open_key_store_service (`

```
hsm_hdl_t session_hdl,
open_svc_key_store_args_t * args,
hsm_hdl_t * key_store_hdl )
```

Open a service flow on the specified key store. Only one key store service can be opened on a given key store.

Parameters

session_hdl	pointer to the handle identifying the current session.
args	pointer to the structure containing the function arguments.
key_store_hdl	pointer to where the key store service flow handle must be written.

Returns

error code.

5.18.5.2 hsm_close_key_store_service() `hsm_err_t hsm_close_key_store_service ( hsm_hdl_t key_store_hdl )`

Close a previously opened key store service flow. The key store is deleted from the HSM local memory, any update not written in the NVM is lost

Parameters

key_store_hdl	handle identifying the key store service flow to be closed.

Returns

error code.

5.19 Life Cycle update

Data Structures

- struct op_lc_update_msg_args_t

Enumerations

- enum hsm_lc_new_state_t {
  HSM_NXP_PROVISIONED_STATE = (1u << 0),
  HSM_OEM_OPEN_STATE = (1u << 1),
  HSM_OEM_CLOSE_STATE = (1u << 3),
  HSM_OEM_FIELD_RET_STATE = (1u << 4),
  HSM_NXP_FIELD_RET_STATE = (1u << 5),
  HSM_OEM_LOCKED_STATE = (1u << 7) }

Functions

- hsm_err_t hsm_lc_update (hsm_hdl_t session_hdl, op_lc_update_msg_args_t *args)

5.19.1 Detailed Description

5.19.2 Data Structure Documentation

5.19.2.1 struct op_lc_update_msg_args_t Structure specifying the life cycle update message arguments

Data Fields

hsm_lc_new_state_t	new_lc_state	

5.19.3 Enumeration Type Documentation

5.19.3.1 hsm_lc_new_state_t `enum hsm_lc_new_state_t`

Enum specifying the Life Cycle state

5.19.4 Function Documentation

5.19.4.1 hsm_lc_update()

```
hsm_err_t hsm_lc_update (
        hsm_hdl_t session_hdl,
        op_lc_update_msg_args_t * args )
```

This API will perform the Life Cycle update

Parameters

session_hdl	handle identifying the session handle.
args	pointer to the structure containing the function arguments.

Returns

error code

5.20 Global Information

Data Structures

- struct global_info_s

Functions

- void populate_global_info (hsm_hdl_t hsm_session_hdl)
- void show_global_info (void)
- uint8_t hsm_get_dev_attest_api_ver (void)
- const char * get_soc_id_str (uint16_t soc_id)
- const char * get_soc_rev_str (uint16_t soc_rev)
- const char * get_soc_lf_str (uint16_t lifecycle)

Variables

- struct global_info_s global_info

5.20.1 Detailed Description

5.20.2 Data Structure Documentation

5.20.2.1 struct global_info_s Global Information structure contain information about SoC and the Library. It will be used globally to take platform specific decisions.

Data Fields

bool	is_populated	to ensure global info is populated once.
uint8_t	ver	Supported version of HSM APIs.
uint16_t	soc_id	SoC ID.
uint16_t	soc_rev	SoC Revision.
uint16_t	lifecycle	Device Lifecycle.
uint32_t	lib_newness_ver	Secure Enclave Library Newness Version.
uint32_t	lib_major_ver	Secure Enclave Library Major Version.
uint32_t	lib_minor_ver	Secure Enclave Library Minor Version.
uint32_t	nvm_newness_ver	NVM Library Newness Version.
uint32_t	nvm_major_ver	NVM Library Major Version.
uint32_t	nvm_minor_ver	NVM Library Minor Version.
char	se_commit_id[GINFO_COMMIT_ID_SZ]	Secure Enclave Build Commit ID.

5.20.3 Function Documentation

5.20.3.1 populate_global_info() `void populate_global_info (`

`hsm_hdl_t hsm_session_hdl )`

This function is called to populate the Global Info structure

Parameters

hsm_session_hdl	identifying the active session.

5.20.3.2 show_global_info() `void show_global_info (`

`void )`

This function prints the Global Information of library

5.20.3.3 hsm_get_dev_attest_api_ver() `uint8_t hsm_get_dev_attest_api_ver (`

`void )`

This function returns the version supported for Device Attestation.

5.20.3.4 get_soc_id_str() `const char* get_soc_id_str (`

`uint16_t soc_id )`

This function returns a string representating SoC ID

Parameters

soc_id	SoC ID fetched from Global Info

Returns

String represention of the SoC ID

5.20.3.5 get_soc_rev_str() `const char* get_soc_rev_str (`

`uint16_t soc_rev )`

This function returns a string representating SoC Revision

Parameters

soc_rev	SoC Revision fetched from Global Info

Returns

String represention of the SoC Revision

5.20.3.6 get_soc_lf_str() `const char* get_soc_lf_str ( uint16_t lifecycle )`

This function returns a string representating Lifecycle

Parameters

lifecycle	value fetched from Global Info

Returns

a string represention of Lifecycle

5.20.4 Variable Documentation

5.20.4.1 global_info `struct global_info_s global_info`

Global Information structure instance which will be populated and later be used for getting the required platform or library details.

5.21 Error codes

Enumerations

- enum hsm_err_t {
  HSM_NO_ERROR = 0x0,
  HSM_INVALID_MESSAGE = 0x1,
  HSM_INVALID_ADDRESS = 0x2,
  HSM_UNKNOWN_ID = 0x3,
  HSM_INVALID_PARAM = 0x4,
  HSM_NVM_ERROR = 0x5,
  HSM_OUT_OF_MEMORY = 0x6,
  HSM_UNKNOWN_HANDLE = 0x7,
  HSM_UNKNOWN_KEY_STORE = 0x8,
  HSM_KEY_STORE_AUTH = 0x9,
  HSM_KEY_STORE_ERROR = 0xA,
  HSM_ID_CONFLICT = 0xB,
  HSM_RNG_NOT_STARTED = 0xC,
  HSM_CMD_NOT_SUPPORTED = 0xD,
  HSM_INVALID_LIFECYCLE = 0xE,
  HSM_KEY_STORE_CONFLICT = 0xF,
  HSM_KEY_STORE_COUNTER = 0x10,
  HSM_FEATURE_NOT_SUPPORTED = 0x11,
  HSM_SELF_TEST_FAILURE = 0x12,
  HSM_NOT_READY_RATING = 0x13,
  HSM_FEATURE_DISABLED = 0x14,
  HSM_KEY_GROUP_FULL = 0x19,
  HSM_CANNOT_RETRIEVE_KEY_GROUP = 0x1A,
  HSM_KEY_NOT_SUPPORTED = 0x1B,
  HSM_CANNOT_DELETE_PERMANENT_KEY = 0x1C,
  HSM_OUT_TOO_SMALL = 0x1D,
  HSM_DATA_ALREADY_RETRIEVED = 0x1F,
  HSM_CRC_CHECK_ERR = 0xB9,
  HSM_OEM_CLOSED_LC_SIGNED_MSG_VERIFICATION_FAIL = 0xF0,
  HSM_OEM_OPEN_LC_SIGNED_MSG_VERIFICATION_FAIL = 0xF0,
  HSM_FATAL_FAILURE = 0x29,
  HSM_SERVICES_DISABLED = 0xF4,
  HSM_UNKNOWN_WARNING = 0xFC,
  HSM_SIGNATURE_INVALID = 0xFD,
  HSM_UNKNOWN_ERROR = 0xFE,
  HSM_GENERAL_ERROR = 0xFF }

5.21.1 Detailed Description

5.21.2 Enumeration Type Documentation

5.21.2.1 hsm_err_t `enum hsm_err_t`

Error codes returned by HSM functions.

Enumerator

HSM_NO_ERROR	Success. The received message is invalid or unknown.
HSM_INVALID_MESSAGE	The provided address is invalid or doesn't respect the API requirements.
HSM_INVALID_ADDRESS	The provided identifier is not known.
HSM_UNKNOWN_ID	One of the parameter provided in the command is invalid.
HSM_INVALID_PARAM	NVM generic issue.
HSM_NVM_ERROR	There is not enough memory to handle the requested operation.
HSM_OUT_OF_MEMORY	Unknown session/service handle.
HSM_UNKNOWN_HANDLE	The key store identified by the provided "key store Id" doesn't exist and the "create" flag is not set.
HSM_UNKNOWN_KEY_STORE	Key store authentication fails.
HSM_KEY_STORE_AUTH	An error occurred in the key store internal processing.
HSM_KEY_STORE_ERROR	An element (key store, key. . .) with the provided ID already exists.
HSM_ID_CONFLICT	The internal RNG is not started.
HSM_RNG_NOT_STARTED	The functionality is not supported for the current session/service/key store configuration.
HSM_CMD_NOT_SUPPORTED	Invalid lifecycle for requested operation.
HSM_INVALID_LIFECYCLE	A key store with the same attributes already exists.
HSM_KEY_STORE_CONFLICT	The current key store reaches the max number of monotonic counter updates, updates are still allowed but monotonic counter will not be blown.
HSM_KEY_STORE_COUNTER	The requested feature is not supported by the firwware.
HSM_FEATURE_NOT_SUPPORTED	Self tests report an issue
HSM_SELF_TEST_FAILURE	The HSM is not ready to handle the current request
HSM_NOT_READY_RATING	The required service/operation is disabled
HSM_FEATURE_DISABLED	Not enough space to store the key in the key group
HSM_KEY_GROUP_FULL	Impossible to retrieve key group
HSM_CANNOT_RETRIEVE_KEY_GROUP	Key not supported
HSM_KEY_NOT_SUPPORTED	Trying to delete a permanent key
HSM_CANNOT_DELETE_PERMANENT_KEY	Output buffer size is too small
HSM_OUT_TOO_SMALL	Data is Read Once, and has already been retrieved
HSM_DATA_ALREADY_RETRIEVED	Command CRC check error
HSM_CRC_CHECK_ERR	In OEM closed lifecycle, Signed message signature verification failure
HSM_OEM_CLOSED_LC_SIGNED_MSG_VERIFI↩ CATION_FAIL	Warning: In OEM open lifecycles, Signed message signature verification failure
HSM_OEM_OPEN_LC_SIGNED_MSG_VERIFIC↩ ATION_FAIL	A fatal failure occurred, the HSM goes in unrecoverable error state not replying to further requests
HSM_FATAL_FAILURE	Message neither handled by ROM nor FW
HSM_SERVICES_DISABLED	Unknown warnings
HSM_UNKNOWN_WARNING	Failure in verification status of operations such as MAC verification, Signature verification.
HSM_SIGNATURE_INVALID	Unknown errors
HSM_UNKNOWN_ERROR	Error in case General Error is received

Index